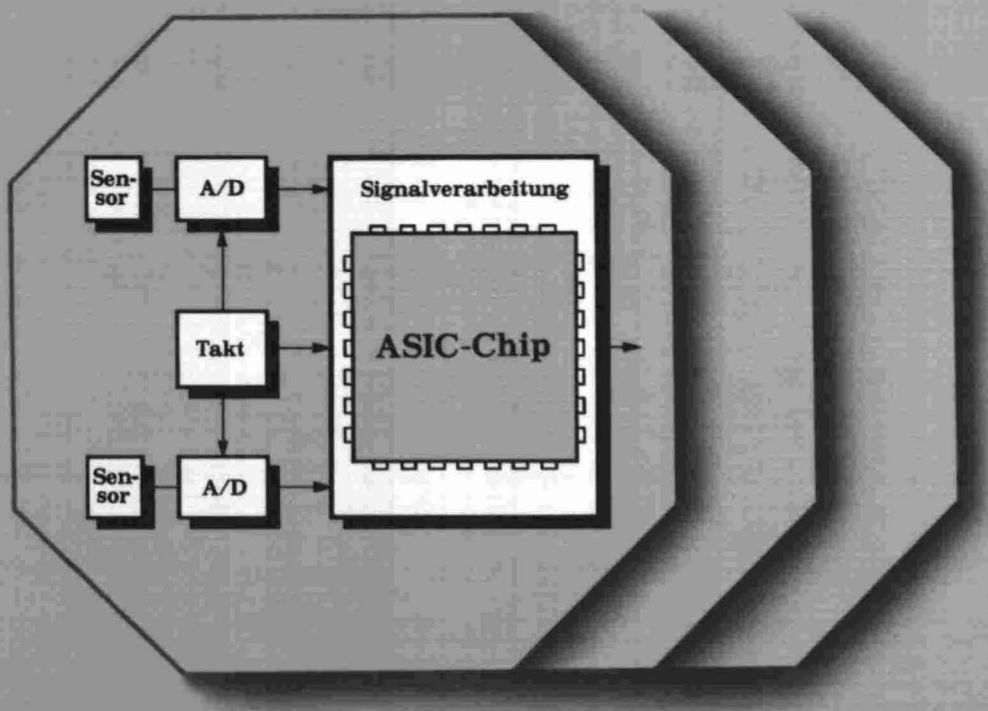
Mikroelektronik

Johann Siegl · Herbert Eichele

Hardware-entwicklung mit ASIC

Einsatz und Anwendung von CAE-Entwurfswerkzeugen



Hüthig

Mikroelektronik

Band 8

Herausgeber
Prof. Dr. Hermann Mader
Prof. Dr. Johann Siegl
Dr.-Ing. Lothar Lerach

In dieser Reihe sind bisher erschienen:

Band 1: H. Reichl, Hybridintegration
Band 2: Kaiser/Hauptmann/Schäfer: Halbleitertechnologie – Stand und Entwicklungstendenzen
Band 3: Auer, Programmierbare Logik-IC
Band 5: Ahlers/Waldmann, Mikroelektronische Sensoren
Band 6: Auer, PLD-Handbuch
Band 7: Fritz, Elektrooptischer Test hochintegrierter CMOS-Schaltungen

Prof. Dr. J. Siegl
Prof. Dr. H. Eichele

Hardwareentwicklung mit ASIC

Einsatz und Anwendung von CAE-Entwurfswerkzeugen

Hüthig Buch Verlag Heidelberg

Prof. Dr. *Johann Siegl* lehrt an der Georg-Simon-Ohm Fachhochschule, Nürnberg, und ist Leiter des ZAM-Anwenderzentrums, Nürnberg.
Prof. Dr. *Herbert Eichele* lehrt an der Georg-Simon-Ohm Fachhochschule, Nürnberg.

CIP-Titelaufnahme der Deutschen Bibliothek

Hardwareentwicklung mit ASIC : Einsatz und Anwendung von CAE-Entwurfswerkzeugen / J. Siegl ; H. Eichele. - Heidelberg : Hüthig, 1990
(Mikroelektronik ; Bd. 8)
ISBN 3-7785-1990-5
NE: Siegl, Johann; Eichele, Herbert; GT

Vorwort

Die Hardwareentwicklung ist im Umbruch begriffen. Schaltungen wurden bislang und werden auch heute noch vielfach durch Versuchsschaltungen in Standard–Technik aufgebaut. An Einzelmodulen erfolgt deren Verifikation sowie die Optimierung einer durch die Spezifikation vorgegebenen Funktion. Der Fortschritt der Halbleitertechnik und der Mikroelektronik erlaubt es Systemfunktionen auf einem Siliziumplättchen anwendungsspezifisch zu integrieren. Neue Techniken, wie PLD–Design, Gate–Array–Design, Cell–Design, Full–Custom–Design haben sich etabliert. Diese neuen Hardware–Realisierungstechniken werden zusammengefaßt unter dem Oberbergiff ASIC: Anwendungsspezifisch Integrierte Schaltungen.

Der Entwurf von Systemfunktionen und deren Realisierung als ASIC auf Silizium erfordert eine systematische Entwurfsmethodik. Wegen der Komplexität und aufgrund der relativ hohen Realisierungskosten ist bei integrierten Techniken eine Verifikation und Optimierung von Schaltungsfunktionen durch "Probieren" ausgeschlossen. Ein Design muß sehr gründlich durchdacht (spezifiziert), verifiziert und optimiert werden, *bevor* es realisiert wird. Moderne Entwurfswerkzeuge auf CAE–Workstations ermöglichen eine systematische Entwurfs– und Verifikationsmethodik.

Das Ziel dieses Buches ist es, in die systematische Entwurfsmethodik und in die Entwurfsverfahren zur Hardwareentwicklung mit ASIC einzuführen. Dies ist vor allem auch eine Einführung in die dafür entwickelten Entwurfswerkzeuge auf CAE–Workstations. Natürlich müssen die neuen Möglichkeiten der Hardware–entwicklung schon jetzt in der Ingenieurausbildung berücksichtigt und gelehrt werden. Das vorliegende Werk ist deshalb als Lehr– und Übungsbuch konzipiert. Es wendet sich an Studierende von hardwareorientierten Studiengängen und an Entwickler in der Praxis, die sich in neue Hardwareentwicklungsmethoden einarbeiten wollen.

Der gesamte Kurs besteht aus drei Modulen. In **Kurs A** erfolgt eine systematische Einführung in den Entwurf Anwendungsspezifisch Integrierter Schaltungen. Der Schwerpunkt liegt hier im Systementwurf und in der Einführung in die Entwurfswerkzeuge. Der **Kurs B** ist ein erster praktischer Vertiefungsteil. Anhand überschaubarer Beispiele werden die heute gängigen Entwurfswerkzeuge geübt bis zur Erstellung einer simulierten Netzliste. In **Kurs C**, dem zweiten Vertiefungsteil wird ein Projektbeispiel behandelt. Der Entwurf wird aufbereitet bis zur Erstellung eines geometrischen Layouts. Es ist dann auch die Postsimulation unter Berücksichtigung des Layouts möglich. Zum Abschluß wird ein Hardwaretest eines realisierten Bausteins durchgeführt.

Die Lehrinhalte der drei Kurse werden in den Kapiteln 1 bis 6 erarbeitet. Die Fragen am Ende eines jeden Kapitels dienen der Erfolgskontrolle, so wie sie in einer mündlichen Prüfung gestellt werden könnten. Das Übungsprogramm ist in Kapitel 7 beschrieben.

Das Kursprogramm, die Beispiele und der Übungsentwurf in Anhang G wurden mit Förderung der Europäischen Gemeinschaft im Rahmen des COMETT–Programms erstellt, entwickelt und praktisch erprobt; dem Förderträger gilt unser Dank. Im Rahmen zahlreicher Diskussionsrunden erhielten wir wertvolle Anregungen von unseren Projektpartnern Herrn Prof. Dr. Doherty, Queens University, Belfast und Herrn Dr. Anderson, University of Ulster, Nordirland. Dank sagen möchten wir auch dem Verlag für die rasche Umsetzung des Manuskripts in Buchform. Unser Dank gilt ferner Herrn Kollegen Prof. Dr. Bleicher, FH München, für seine Anregungen. Ganz besonders danken möchten wir Herrn Dipl.–Ing. (FH) Haspel für sein Engagement und für seine Unterstützung bei der Ausarbeitung von Beispielen. Herr Haspel hat sehr viele wertvolle Details zur Einführung in die Entwurfswerkzeuge beigetragen.

Nürnberg, im Juni 1990

Inhaltsverzeichnis . . . . . Seite

1. Einführung

Die Mikroelektronik weist eine sehr stürmische Entwicklung auf. Am Anfang der Elektronikentwicklung stand die Röhre als Verstärkerelement. Wegen des relativ voluminösen Aufbaus hatten damit bestückte Elektronikgeräte meist weniger als 10 Verstärkerelemente. Seit der Erfindung des Transistors 1948 und insbesondere nach Einführung des integrierbaren planaren Transistors etwa im Jahre 1960 ist die Komplexität elektronischer Systeme ebenso rasch angestiegen wie der Integrationsgrad mikroelektronischer Bausteine. Vor 25 Jahren bestand ein Elektronik-Produkt zumeist aus weniger als 1000 Transistorelementen. 40 Jahre nach Erfindung des Transistors sind auf dem Markt eingeführte Elektroniksysteme beispielsweise aufgebaut aus 16- oder 32-Bit-Mikrorechnern und 256K-RAM-Speichern, so daß diese Systeme mehr als 10 Millionen Transistorfunktionen enthalten. Die Erhöhung der Komplexität von Elektroniksystemen ist noch nicht abgeschlossen. Heute ist der 4M-Bit-Speicherbaustein bereits in Stückzahlen verfügbar; die Entwicklung zeigt Bild 1.1. Derzeitige Forschungsanstrengungen konzentrieren sich auf den 64M-Bit-Speicherbaustein mit Strukturgeometrien im Zehntel-µm-Bereich.

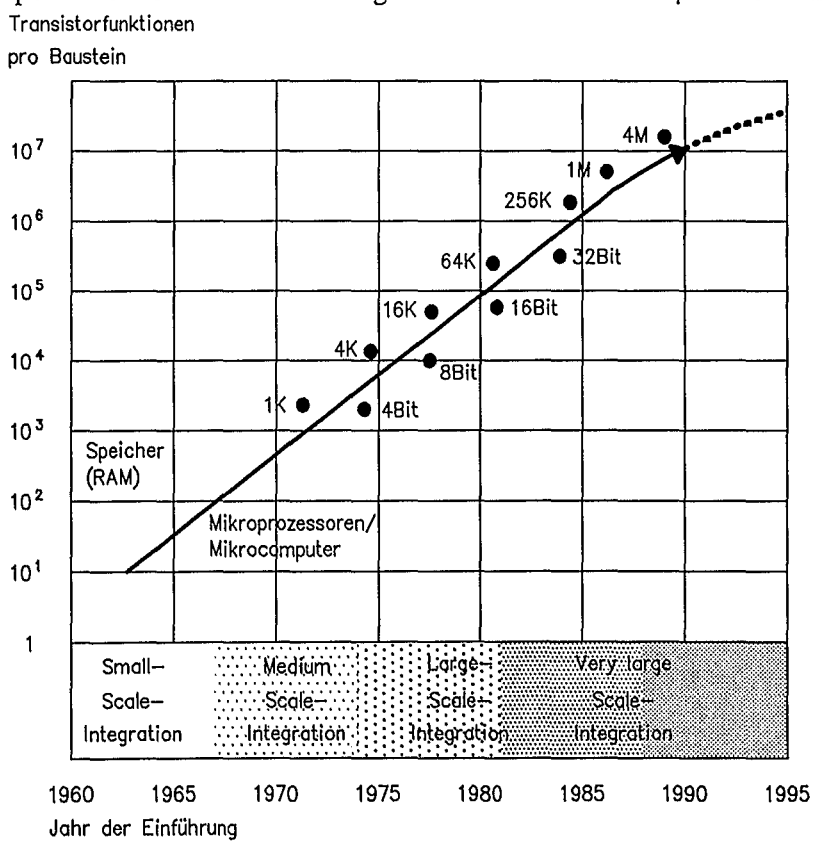


Bild 1.1: Entwicklung der Schaltungsintegration

Parallel zu dem technologischen Fortschritt in den letzten 20 Jahren hat sich die Fähigkeit des Entwurfs-Ingenieurs, derart komplexe Systeme zu entwerfen, ebenso weiterentwickelt. Der Bereich Computer-Aided-Engineering in der Elektronikentwicklung ist heute ein sehr rasch sich ausweitendes Gebiet. Die CAE-Werkzeuge ermöglichen dem Entwurfs-Ingenieur die Lösung außerordentlich komplexer Designaufgaben in relativ kurzer Zeit. Als Folge davon ergeben sich immer kürzer werdende Produktlebensdauern mit kürzeren Entwicklungszeiten. Der Entwurfs-Ingenieur löst seine Entwicklungsaufgaben zunehmend mit geeigneten CAE-Werkzeugen an einer CAE-Workstation (Bild 1.2). Diese CAE-Werkzeuge sind für ihn ein virtuelles Labor mit sehr vielfältigen Test- und Verifikationsmöglichkeiten während der Entstehung des Entwurfs. Neben den Entwurfswerkzeugen werden technologieabhängige Bauteilbibliotheken benötigt. Die Entwurfswerkzeuge selbst lassen sich einteilen in Werkzeuge zur Festlegung der Schaltungsstruktur; Werkzeuge zur Schaltungsverifikation (Simulation); Werkzeuge zur Layouterstellung.

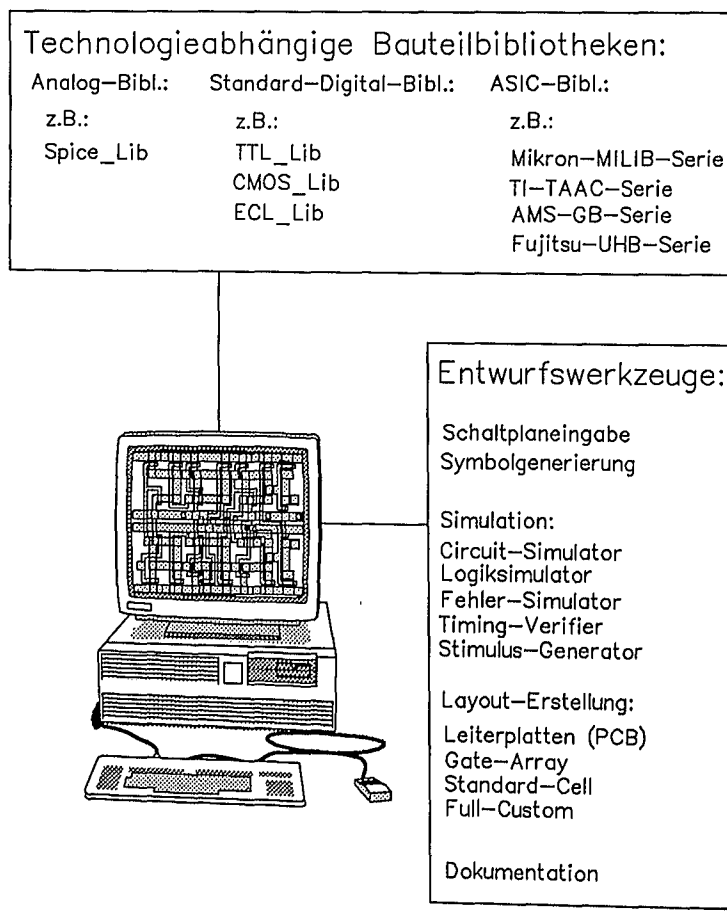


Bild 1.2: Entwurfsunterstützung durch Entwurfswerkzeuge auf CAE-Workstations

Die Entwicklung von Elektroniksystemen vollzieht sich in mehreren Phasen. Bild 1.3 zeigt die Entwicklungsphasen eines Elektronik-Produktes von der Produktidee über die Systemkonzeption, den Subsystementwurf, die technologische Realisierung, die Aufbautechnik, die Testphase bis schließlich zur Systemintegrationsphase. Für die Systementwurfsphase, die Subsystementwicklungsphase und die zugehörige Layouterstellung zur Vorbereitung der technologischen Realisierung gibt es heute geeignete Entwurfswerkzeuge an integrierten Entwurfssystemen. Dabei werden beispielsweise bei der Schaltungsverifikation durch Simulation bereits die Testvektoren verwendet, wie sie hinterher der Tester für den Hardwaretest eines Funktionsmoduls benötigt.

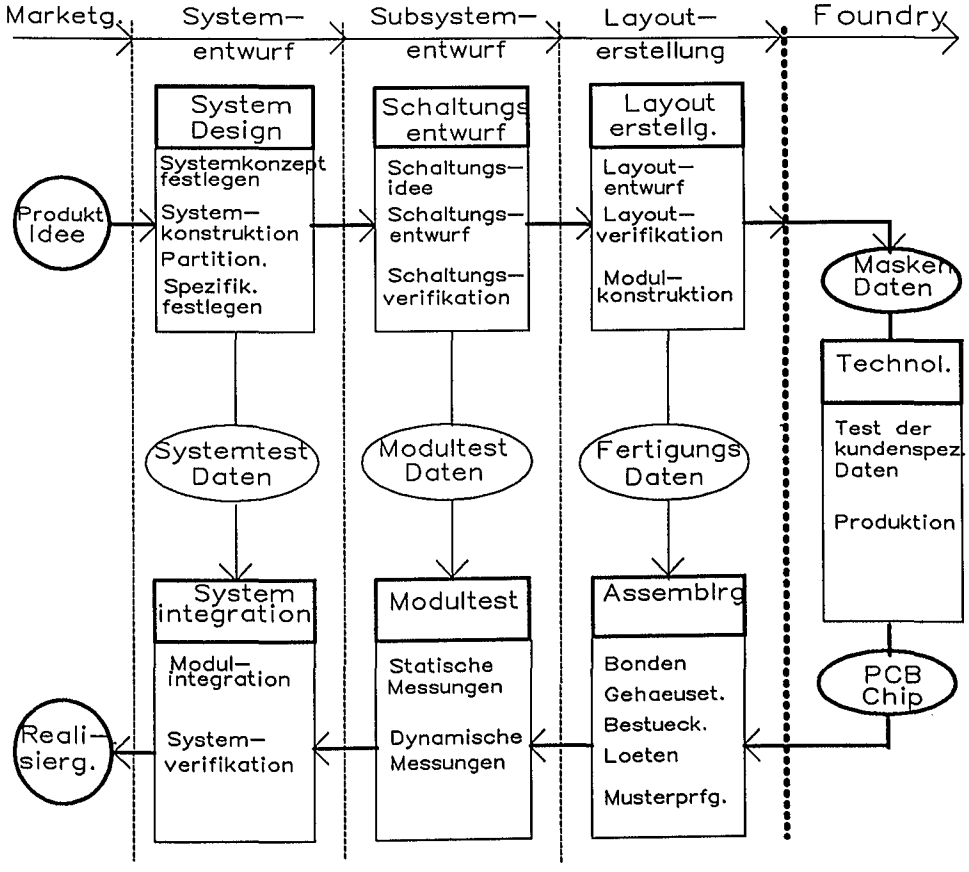


Bild 1.3: Entwurfsphasen bei der Entwicklung eines Elektroniksystems

Ergebnis des Systementwurfs in der ersten Entwicklungsphase ist die Aufteilung des Systems in Funktions–Module und deren genaue Spezifikation. Ausgehend von der Spezifikation werden in der zweiten Phase die Funktions–Module unter Berücksichtigung der technologischen Realisierung entwickelt. Die prinzipielle Vorgehensweise zeigt Bild 1.4. Der Schaltungsentwickler kann auf bekannte und bewährte Funktionsbausteine, die ihm die ausgewählte Schaltkreistechnik zur Verfügung stellt zurückgreifen. Durch geeignetes Zusammenfügen von Funktions–bausteinen löst er die gestellte Entwicklungsaufgabe. Dies ist die eigentliche Ingenieuraufgabe, wo Kreativität, Ideenreichtum und hinreichende Kenntnisse der Schaltkreistechnik erforderlich sind. Ziel ist es, durch eine umfassende Entwurfsverifikation ein Redesign zu vermeiden. Vor der technologischen Realisierung sollte daher der Entwurf soweit wie möglich verifiziert werden.

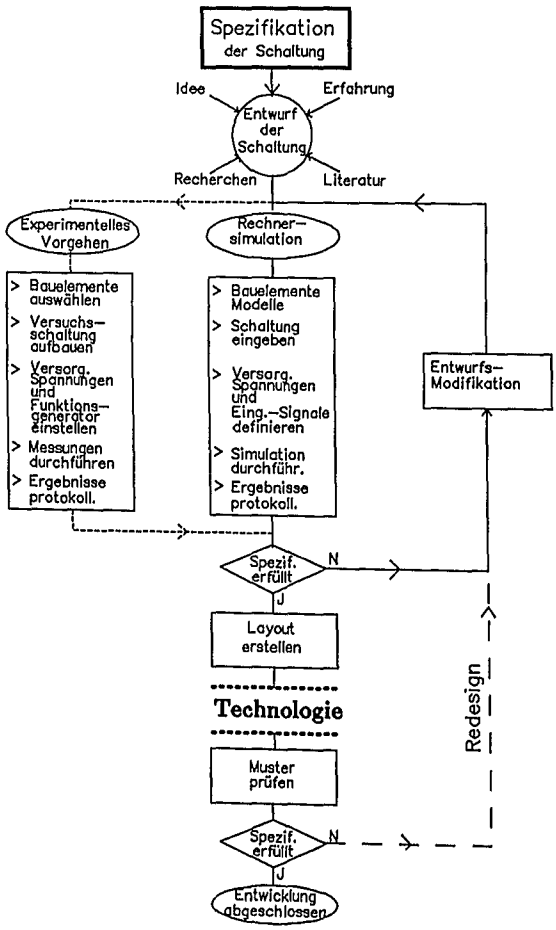


Bild 1.4: Prinzipielle Vorgehensweise bei der Hardwareentwicklung von Funktionsbausteinen

Die verschiedenen Realisierungstechniken eines Elektroniksystems sind in Bild 1.5 zusammengefaßt. Man unterscheidet grundsätzlich die Softwarelösung von der Hardwarelösung. Bei der Softwarelösung sind als Bausteine Standardprozessoren, Signalprozessoren, Speicherbausteine und Peripheriebausteine verfügbar. Diese Bausteine sind allesamt vorgefertigt. Die anwenderspezifische Lösung liegt in dem für das jeweilige Subsystem zu entwickelnden Programm für den Mikroprozessor.

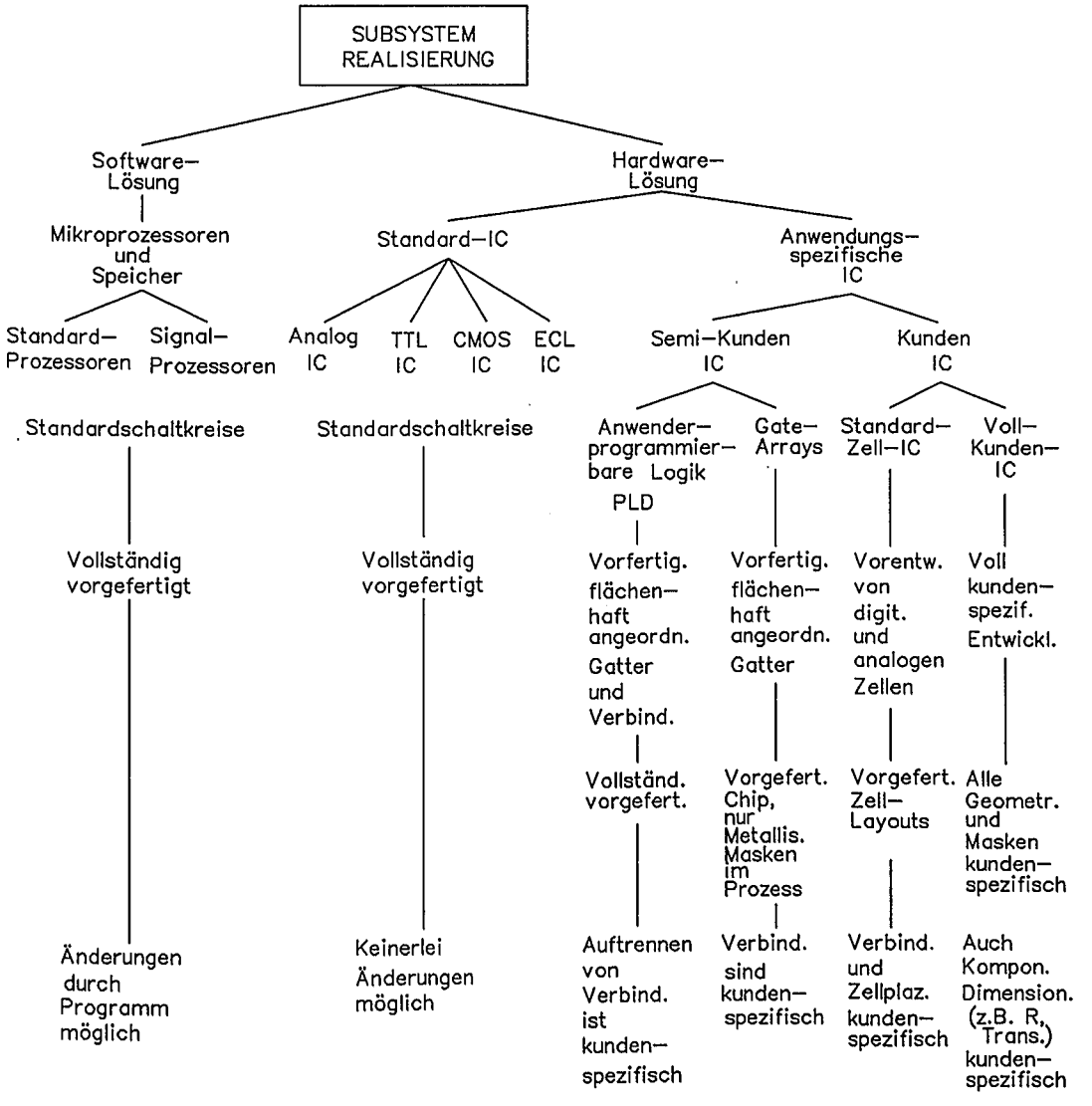


Bild 1.5: Übersicht der möglichen Hardware–Realisierungstechniken

Bei der Hardwarelösung unterscheidet man die Realisierung mit Standard-IC und die Realisierung mit Anwendungsspezifisch Integrierten Bausteinen (ASIC: Gate-Arrays; Standard-Zell-IC; Voll-Kunden-IC). Die Standard-IC lassen sich wiederum in Analog-IC (z.B. Operationsverstärker; Analog-Digitalwandler; u.a.) und Digital-IC einteilen. Bei Digital-IC stehen heute mehrere Schaltkreisfamilien (u.a. TTL; CMOS; ECL) mit einer großen Vielfalt von Funktionsbausteinen zur Verfügung. Die Anwendungsspezifisch Integrierten IC werden in Semi-Kunden-IC und Kunden-IC eingeteilt. Allgemein sind ASIC-Schaltkreise wegen vielfältiger Vorteile sehr stark im Kommen. Als Beispiel für einen Anwendungsspezifisch Integrierten Baustein ist in Bild 1.6 ein Gate-Array dargestellt. Die Siliziumfläche besteht aus Input/Output-Zellen am Rand außen und den Basiszell-Reihen. Die Basiszell-Reihen enthalten Transistor-Arrays. Durch anwendungsspezifische Verbindung der Transistorelemente in den vorgefertigten Basiszell-Reihen entstehen verschiedene Funktionsschaltungen bzw. Makrozellen (z.B. NAND; D-Flipflop; Schieberegister; u.a.). Die Verdrahtungsinformation für die Makrozellen wird aus einer prozeßspezifischen Makrozellen-Bibliothek entnommen. In den zwischen den Basiszell-Reihen befindlichen Verdrahtungskanälen können die Makrozellen anwendungsspezifisch miteinander verbunden werden. Üblich bei Gate-Arrays sind zwei anwendungsspezifische metallische Verdrahtungsebenen.

Im Gegensatz zu Gate-Arrays sind die Standardzellen-IC nicht vorgefertigt. Die Standardzell-Bibliothek enthält vorentwickelte Makrozellen mit allen für den Entwurf notwendigen Daten (u.a. Symbole, Simulationsdaten, Geometrien). Die Makrozellen haben eine einheitliche Höhe, sind aber hinsichtlich der Anordnung verschieblich. Dies ermöglicht einen flexibleren Einsatz und damit einen höheren Ausnutzungsgrad der Siliziumfläche. Bei der Herstellung sind jedoch mehr Verarbeitungsschritte und auch mehr Maskensätze erforderlich. Während bei Fullcustom-Bausteinen keine Vorfertigung vorliegt, ist am unteren Ende der ASIC-Skala bei den PLD alles vorgefertigt.

Die programmierbaren Logikbausteine (PLD) enthalten zwei hintereinander-geschaltete Verknüpfungsfelder, die als logische UND- bzw. ODER- Ebene bezeichnet werden. Die Verknüpfungsfelder sind programmierbar, ähnlich wie bei speicherprogrammierbaren Bausteinen (PROM bzw. EPROM). Auf der Grundlage höchst regulärer Transistorstrukturen ist damit eine Realisierung kombinatorischer Logik möglich. In Verbindung mit Registerblöcken, deren Ausgänge u.a. auch auf die Eingangsbeschaltung zurückgeführt werden können, sind auch sequentielle Schaltungen realisierbar.

Je mehr kundenspezifische Masken erforderlich sind, umso kostspieliger ist ein Redesign. Bei kundenspezifisch integrierten Schaltkreistechniken sind Redesigns außerordentlich kostspielig und zeitraubend. Der Entwurf von ASIC-Schaltkreisen macht daher den Einsatz von CAE-Tools zwingend notwendig.

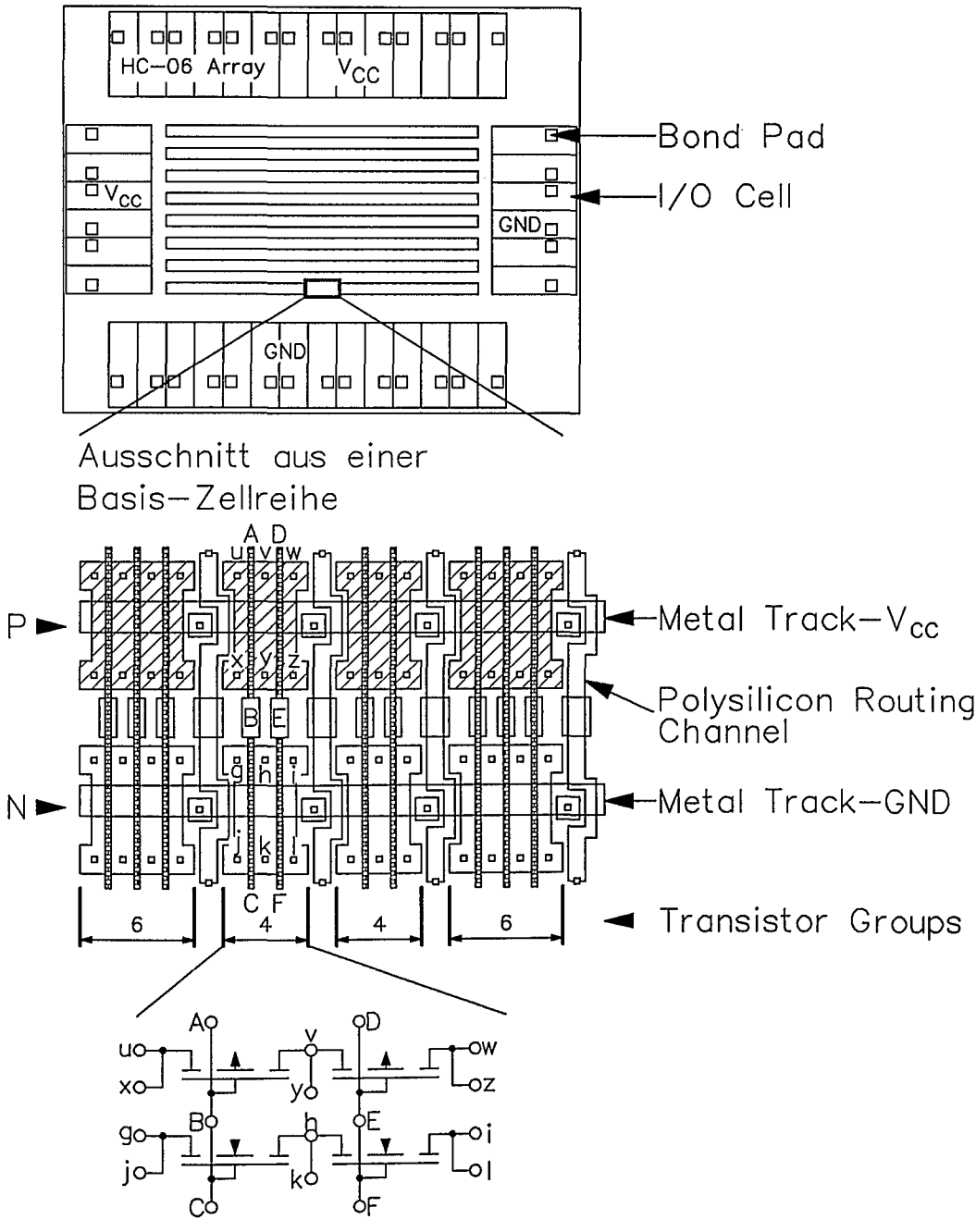


Bild 1.6: Prinzipieller Aufbau eines Gate–Array–Bausteins

Die Vorteile der Anwendungsspezifisch Integrierten Bausteine sind u.a.:

* geringere Anzahl der Bestückungselemente; weniger Verbindungselemente; geringerer Flächenbedarf; geringeres Bauvolumen eines Gerätes.

* höhere Signalverarbeitungsgeschwindigkeit; geringere Leistungsaufnahme.

* höhere Zuverlässigkeit; komplexere Funktionsteile automatisch testbar; Kontrollfunktionen leicht realisierbar.

* Know-how Schutz.

Zur Veranschaulichung des Flächenbedarfs: Mit Standard-IC können je nach Komplexität der ausgewählten IC ca. 2000 Gatterfunktionen auf einer Europakarte (Größe: 100 mm * 160 mm) untergebracht werden. Diese Zahl mag verdeutlichen, welch hohe Packungsdichten bei Anwendungsspezifisch Integrierten Bausteinen mit bis zu 100.000 Gatteräquivalenten erzielt werden können.

Der wirtschaftliche Einsatz der verschiedenen anwendungsspezifisch integrierten Techniken orientiert sich nach Stückzahl und Preis. Je höher der Grad der Vorfertigung, um so schlechter ist der Ausnutzungsgrad der Siliziumfläche. Je geringer der Grad der Vorfertigung, um so höher sind die Entwicklungskosten. Eine Übersicht über den Einsatzbereich der unterschiedlichen Entwurfsverfahren zeigt Bild 1.7. In Tabelle 1.1 sind wichtige Merkmale der ASIC-Entwurfsverfahren zusammengestellt.

Ein komplexes Elektroniksystem besteht oftmals aus verschiedenen Funktionsmodulen, die in unterschiedlichen Entwurfsverfahren realisiert werden. Die Funktionsmodule werden dann meist auf Leiterplatten aufgebracht und verbunden (u.a. Multichip-on-Board). So kann beispielsweise ein Sensorsignal mit einem Anwendungspezifisch Integrierten Baustein aufbereitet und nachfolgend mit einem Standard-Mikroprozessorsystem verarbeitet werden. Es ist deshalb notwendig, daß der Entwurfs-Ingenieur möglichst alle in Bild 1.5 skizzierten Realisierungstechniken und deren jeweilige Entwurfstechnik beherrscht, um zu einer optimalen Produktlösung zu kommen. Der Entwurfs-Ingenieur muß als "long thin man" Kenntnisse der Schaltkreistechnologie, der Schaltungstechnik, der Aufbautechnik, der Systemtechnik und der Prüftechnik mitbringen. Wichtig für ihn ist das Beherrschen der Entwurfswerkzeuge auf den CAE-Workstations. Jeder Elektronik-Entwickler sollte um die Möglichkeiten der CAE-Tools für seinen Bereich wissen, damit er selbst einschätzen kann, wo und in welcher Form eine Unterstützung für ihn möglich ist. Die nachstehenden Ausführungen sollen in die Entwurfsmethodik und den Einsatz von CAE-Tools für den Entwurf Anwendungsspezifisch Integrierter Schaltungen einführen.

Tabelle 1.1: Zum Vergleich der unterschiedlichen Entwurfsverfahren

	PLD	Gate–Array	Standard–Cell	Full–Costum
Gate–Komplexität	100...5000	1000...50000	10000...200000	>200000
Entwicklungs–zeit	1..3 Wochen	4..20 Wochen	5..50 Wochen	1 bis 2 Jahre
Wirtschaftl. Einsatzbereich in Stückzahlen pro Jahr	bis ca. 1000	ca. 1000 bis ca. 10000	ca. 10000 bis ca. 100000	> ca. 100000
Maskensätze	0	2	mehrere	viele

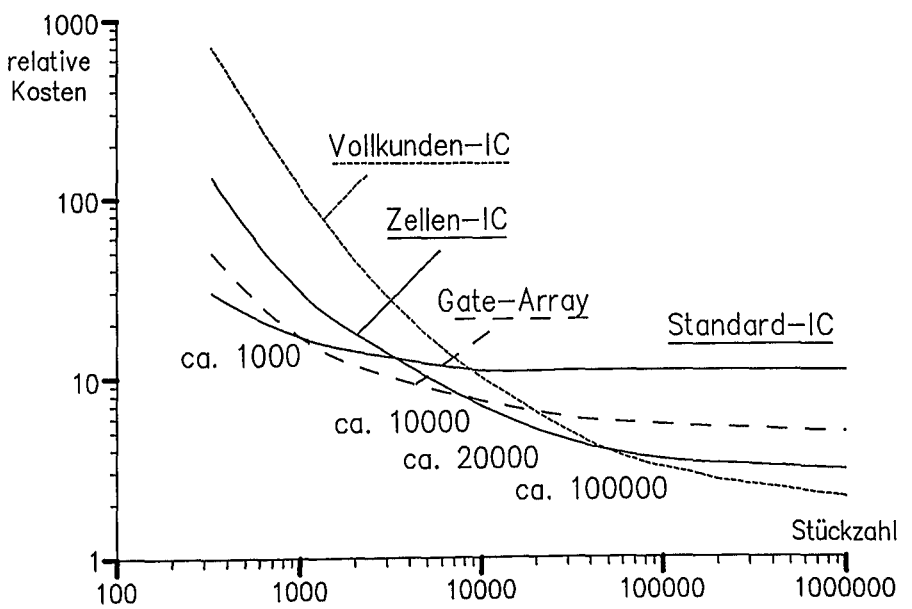


Bild 1.7: Zum wirtschaftlichen Einsatz Anwendungsspezifisch Integrierter Techniken

Literatur zu Kapitel 1

/1/ E. Göttler, L. Haschik, E. Hörbst, G. Sandweg et al.: Entwicklung von kundenspezifischen Schaltungen. Elektronik, Hefte 19,20,21,22, 1984

/2/ A. Glasser, D. W. Dobberpuhl: The Design and Analysis of VLSI Circuits. Addison-Wesley, 1985

/3/ E. Hörbst, M. Nett, H. Schwärtzel: VENUS-Entwurf von VLSI-Schaltungen. Springer, 1986

/4/ B. Höfflinger (Hrsg.): Großintegration. Oldenbourg, 1979

/5/ J. Millman: Microelectronics. McGraw Hill, 1979

/6/ H. Weinerth: Schlüsseltechnologie Mikroelektronik. Elektronik; Beitragsserie: 26 Beiträge im Jahr 1989

/7/ Sonderheft ASIC's; Elektronik 1988

/8/ W. Rosenstiel, R. Camposano: Entwurf hochintegrierter MOS-Schaltungen. Springer, 1989

/9/ A. Rappaport: A Designers Guide to Semicustoms Design. Deutsche Übersetzung: Erfahrungen mit Gate-Arrays. te-wi-Verlag, 1985

Fragen zu Kapitel 1

F1.1 Nennen Sie die Standard–Entwurfswerkzeuge auf einer CAE–Workstation! In welcher Entwurfsphase werden sie eingesetzt?

F1.2 Welche Entwurfsphasen werden bei der Entwicklung eines Elektroniksystems durchlaufen?

F1.3 Beschreiben Sie die Vorgehensweise bei der Hardwareentwicklung von Schaltungsmodulen!

F1.4 Welche Möglichkeiten zur Realisierung einer Logikfunktion sind heute Stand der Technik und wodurch unterscheiden sie sich?

F1.5 Skizzieren Sie den prinzipiellen Aufbau eines Gate–Array–Bausteins!

F1.6 Nennen Sie die Vorteile für den Einsatz von ASIC!

F1.7 Nennen Sie den Einsatzbereich von PLD bzw. von Gate–Arrays!

F1.8 Wodurch unterscheidet sich ein Standard–Zell–Design von einem Gate–Array–Design?

2. Zum Systementwurf

2.1 Aufgabenbeschreibung durch ein Pflichtenheft

Der prinzipielle Aufbau eines Elektroniksystems ist in Bild 2.1 dargestellt. Ein derartiges System besteht aus folgenden Funktionseinheiten:

* Sensoren bzw. Empfänger
* Aktoren bzw. Sender
* A/D und D/A-Wandler
* Digitalteil
* Analogteil
* Schnittstellenmodule
* Energieversorgung

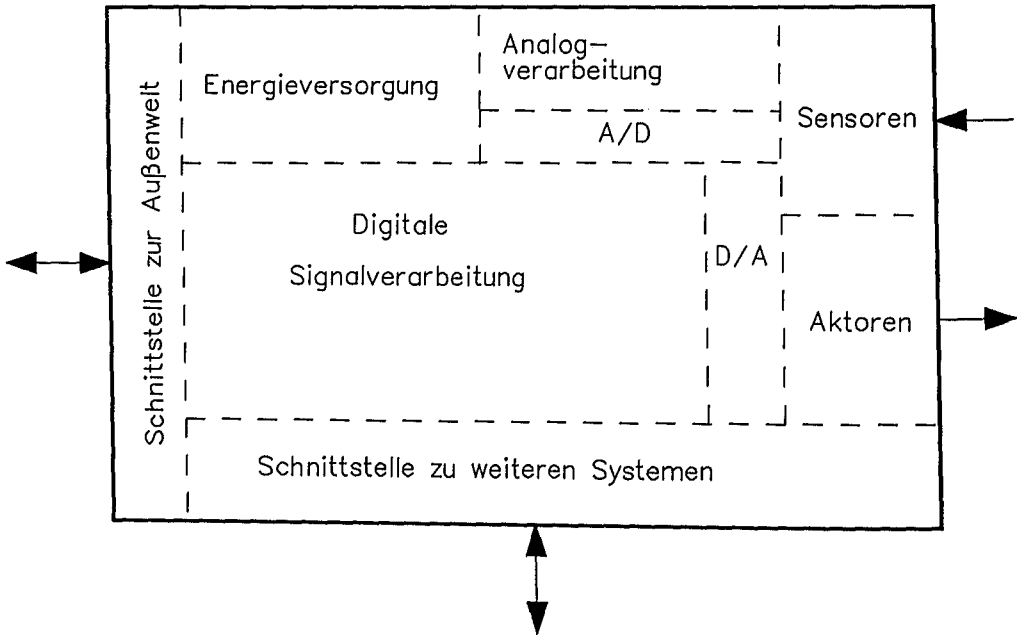


Bild 2.1: Zum Aufbau eines Elektroniksystems

Bild 2.2 zeigt als Beispiel eine Bildverarbeitungskarte als Zusatz–Steckkarte für einen Personalcomputer. In diesem Fall ist der Sensor eine CCD–Kamera. Die Kamera liefert ein analoges Bildsignal. Im Analogteil wird das Bildsignal vorverarbeitet und digitalisiert. Im Digitalteil erfolgt die digitale Weiterverarbeitung des Bildsignals; konkret kann z.B. hardwaremäßig zeilenweise ein Lauf–Längencode gebildet und in einem Pufferspeicher zwischengespeichert werden. Der Busmodul steuert die Übergabe der vorverarbeiteten Bildsignaldaten an das nächste System, den Personalcomputer.

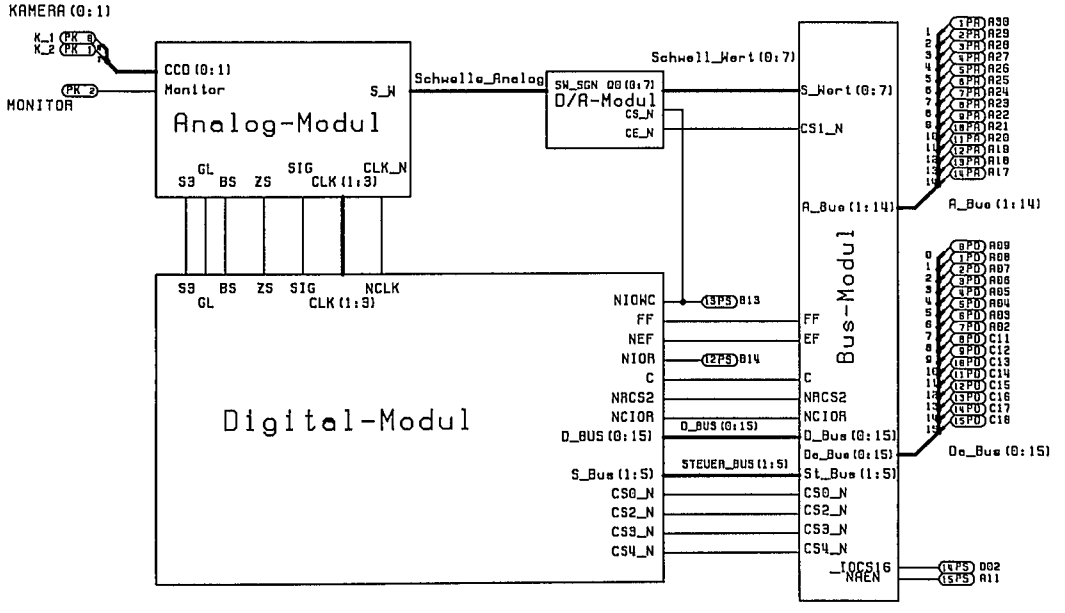


Bild 2.2: Beispiel eines realisierten Elektroniksystems

Das Pflichtenheft hierzu sollte mindestens folgende Punkte beinhalten:

* Ziel und Zweck des Projektes
* Gesamtbeschreibung
* Funktionelle Anforderungen
* Anforderungen an Systemschnittstellen
* Leistungsanforderungen
* Einschränkungen für den Entwurf
* Geforderte Qualitäten und Eigenschaften
* Prüf– und Abnahmebedingungen
* Energieversorgung
* Umgebungsbedingungen
* Sonstige Anforderungen

Vor der Festlegung des Pflichtenheftes ist im allgemeinen eine Machbarkeitsstudie erforderlich. Das Pflichtenheft ist ein Anforderungsdokument für den Abnehmer eines Produktes. Am Beispiel der Bildverarbeitungskarte soll ein Pflichtenheft in Kurzform vorgestellt werden:

Ziel und Zweck: Zweidimensionale Objekt- bzw. Schriftenerfassung mittels CCD-Kamera zur Objekt- bzw. Schriftenerkennung und anschließender Qualitätskontrolle mittels Personalcomputer.

Gesamtbeschreibung: Es ist eine Einsteckkarte in IBM-kompatible AT-Personalcomputer zu entwickeln. Die Einsteckkarte bewerkstelligt eine Bildsignalvorverarbeitung des von der CCD-Kamera gelieferten Videosignals zur Mustererkennung am Personalcomputer.

Funktionelle Anforderungen: Das an einem Monitor beobachtbare Videosignal ist zu binarisieren bei einstellbarer, vom PC aus konfigurierbarer Komparatorschwelle. Vom binarisierten Videosignal ist zeilenweise in Realtime ein Lauf-Längencode zu ermitteln und in einem Stapelspeicher abzulegen. Der PC soll über ein Bus-Modul an den Stapelspeicher zugreifen können. Nach einer Vollbildaufnahme mit maximal 5.000 Hell/Dunkelwechseln und der Ermittlung von Bezugszeilen sollen konfigurierbare Prüfzeilen einstellbar sein.

Anforderungen an Systemschnittstellen: Schnittstelle zur CCD-Kamera: vorgegeben durch ausgewählte Kamera (512 * 512 Bildpunkte). Schnittstelle zum Monitor: Videonorm. Schnittstelle zum Personalcomputer: AT-Bus. Zur Ansteuerung der BV-Karte vom PC aus ist ein Software-Treiber zu entwickeln, so daß bestimmte Grundfunktionen (Vollbildaufnahme, Bildbereichsaufnahme, Prüfzeilen setzen, Schwellwert setzen, Retransmit-Funktion, u.a.) als Makro von einer höheren Programmiersprache aus aufrufbar sind.

Leistungsanforderungen: Als erste Applikation soll die Erkennung und Auswertung von LCD-Anzeigen, Siebensegmentanzeigen und gemultiplexten Fluroszenzanzeigen realisiert werden. Folgende Merkmale sind dabei zu beachten:

* Lageunabhängige Displayerkennung;
* Erkennung des Nebenleuchtens bei 5 % der Normalhelligkeit;
* Einlesen und erkennen einer 4 stelligen Zahl innerhalb <100ms.

Einschränkungen für den Entwurf: Verwendung von Standard-Bausteinen und PLD wegen der geringen zu erwartenden Stückzahlen; Aufbau als PCB; Software-Treiber in C.

Geforderte Qualitäten und Eigenschaften: Elektrische Sicherheit nach VDE; betriebsicherer Aufbau; Erstellung eines Prüfprogramms; vollständige Dokumentation der Hardware und Software.

Energieversorgung: +/- 12V; +/- 5V; GROUND versorgt über AT-Slot.

Umgebungsbedingungen: Hinsichtlich Umgebungstemperatur, Störstrahlungsfestigkeit, EMV-Verträglichkeit sind die gleichen Anforderungen wie für einen Personalcomputer zu erfüllen.

Sonstiges: Fragen der Zuverlässigkeit, Garantie, Wartung und Service werden festgelegt nach Entwicklung und Erprobung von 5 Prototypen.

Die Systemspezifikation ist eine Festlegung für den Entwerfer. Bei komplexeren Systemen sind mehrere ausführende Stellen bei der Entwurfsrealisierung beteiligt. Deshalb ist es notwendig, die Entwurfsbedingungen und Anforderungen umfassend festzulegen. Allgemein enthält die Systemspezifikation detaillierte Spezifikationsdaten, sie lassen sich in folgende Bereiche unterteilen:

Schnittstellenbeschreibung:
* Definition der Interaktionsstellen (Ports, Pins, u.a.)
* Signalformen an den Interaktionsstellen
* Absolute Grenzwerte und Aussteuerbedingungen
* Kritische Signalpfade
* Signalübertragungsanforderungen
* Toleranzen
* Gehäuse, Aufbau- und Verbindungstechnik

Verhaltensbeschreibung:
* Funktionsdefinition
* Eingangs-/Ausgangsbeziehungen
* Simulationsmodelle
* Algorithmen, Übertragungsfunktionen

Strukturbeschreibung:
* Definition von Teilfunktionen; Hierarchiedarstellung
* Verbindung von Gesamt- und Teilsystem
* Strategie bezüglich Testbarkeit

Systemumgebung:
* Temperatur, mechanische Belastungen, chemische Belastungen,
* Strahlungsbelastungen
* Anforderungen an die Elektromagnetische Verträglichkeit
* Anforderungen an die Zuverlässigkeit

Testumgebung:
* Testsysteme, Testsystemverbindungen
* Sicherheitsgrenzen
* Produktionstest, Funktionstest
* Qualitätssicherungsmaßnahmen

Entwurfsmanagement:
* Projektierung, Projektablaufplan
* Einsatzplanung von Personal und Entwurfshilfsmitteln
* Preis/Leistungsbewertung
* Dokumentation

2.2 Ansätze zur Entwurfsautomatisierung

2.2.1 Aufgaben und Ziele der Entwurfsautomatisierung

Bild 2.3 zeigt die Umsetzung der Anforderungsspezifikation in einen Systementwurf. Danach erfolgt schließlich die Systemimplementierung. Für den Systementwurf gibt es verschiedene Hilfsmittel und Werkzeuge als Ansatz zur Entwurfsformalisierung und Entwurfsautomatisierung.

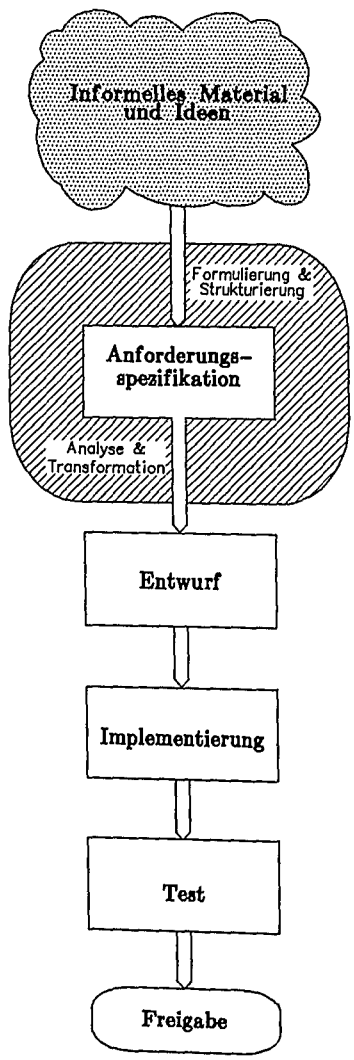


Bild 2.3: Von der Anforderungsspezifikation zum Entwurf und zur Implementierung

Unter Entwurfsautomatisierung versteht man die Generierung (Synthese), Verifikation und automatische Datenhaltung (in Datenbanken) von Entwürfen in den verschiedenen Entwurfsebenen. Zunehmend erstellen geeignete Werkzeuge automatisch Teile eines Entwurfs bzw. es werden Entwurfsschritte automatisiert. Die Entwurfsautomatisierung hat folgende Ziele:

* Reduzierung der Entwurfs- und Entwicklungszeiten;
* weitgehendes Ausschließen von Entwurfsfehlern;
* Vereinfachung von Änderungen in der Entwurfsphase;
* Überprüfbarkeit eines Entwurfs;
* Ermöglichung und Vereinfachung des Tests (Design for Testability).

Bei einem kundenspezifischen Gate-Array mit 10.000 Gatterfunktionen macht der Entwurf einen Großteil der anfallenden Kosten aus. Ein manueller Entwurf ist hier in vertretbarer Zeit nicht mehr darstellbar. Mit heute verfügbaren Werkzeugen zur Entwurfsautomatisierung gelingt es, außerordentlich komplexe Systeme in relativ kurzer Zeit zu entwerfen. Beim Entwurf komplexerer Bausteine, mit z.B. 50.000 Gatterfunktionen beansprucht die Konzeptionsphase mehr als 50% der Entwicklungszeit. Ganz besonders wichig ist dabei die Berücksichtigung der Anforderungen an die Testbarkeit durch besondere Testhilfen bereits auf Architekturebene.

Für ein weiterführendes Studium der in diesem Kapitel angeschnittenen Thematik der Entwurfsautomatisierung sei auf /2/, /3/ verwiesen.

2.2.2 Hierarchisches Konzept und Funktionsspezifikation von Teilfunktionen

Ziel der Entwurfsmethodik ist es, die Komplexität eines Systems durch ein modulares Konzept zu reduzieren und damit die Überprüfung der Korrektheit zu erleichtern. Diese Vorgehensweise wird mit dem Schlagwort "Strukturierter Entwurf" gekennzeichnet. Erstmals propagierten u.a. Mead und Conway /6/ diesen Entwurfsstil. Ziel der Anstrengungen ist es, soweit wie möglich Synthesewerkzeuge beim Entwurf einzusetzen. Im Idealfall sollte ein Funktionsmodul ausgehend von der Funktionsspezifikation automatisch in eine Schaltungsstruktur umgesetzt werden. Diese Synthese einer Schaltungsstruktur ist optimal nur in Teilbereichen möglich. Leichter automatisierbar ist die Umsetzung einer symbolisch beschriebenen Schaltungsstruktur in ein geometrisches Layout. Mit Silicon-Compilern haben Systementwickler die Möglichkeit ausgehend von einer herstellerunabhängigen Beschreibung den Entwurf weitgehend automatisch umzusetzen in ein Layout auf Silizium. Damit wird die Umsetzung des Entwurfs in Silizium für den Entwerfer erleichtert. Eine wichtige Basis für eine herstellerunabhängige Entwurfsbeschreibung sind geeignete Hardwarebeschreibungssprachen (siehe Abschnitt 2.5.3). Zur Vertiefung der Thematik über Silicon-Compiler sei auf /9/, /10/ verwiesen.

Bild 2.4 verdeutlicht nach dem Phasenmodell die einzelnen Synthesephasen von der Problemspezifikation bis zur Chiprealisierung.

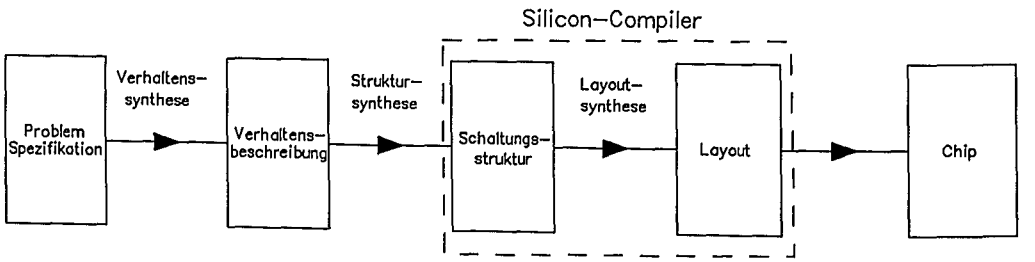


Bild 2.4: Die Entwurfsphasen mit und ohne Silicon-Compiler

In den letzten Jahren wurden für die Lösung komplexer Softwareaufgaben sehr effiziente Techniken entwickelt. Unter Ausnutzung eines streng hierarchischen Konzeptes und der damit gegebenen Modularität können Techniken, wie sie für Softwareaufgaben entwickelt wurden auch teilweise für den Hardwareentwurf übernommen werden. Der wesentliche Unterschied zu Softwareaufgaben besteht darin, daß beim Hardwareentwurf schon in der Systemkonzeptphase die physikalischen Beschränkungen durch die Implementierung berücksichtigt werden müssen. Bild 2.5 zeigt schematisch wie durch Regelüberprüfungen (Restriktionen; Gültigkeitsbereich) ein Lösungsvorschlag prinzipiell generiert werden kann. Eine systematische Darstellung des Hardware-Entwurfs von digitalen Systemen ist u.a. in /3/ ... /8/ beschrieben.

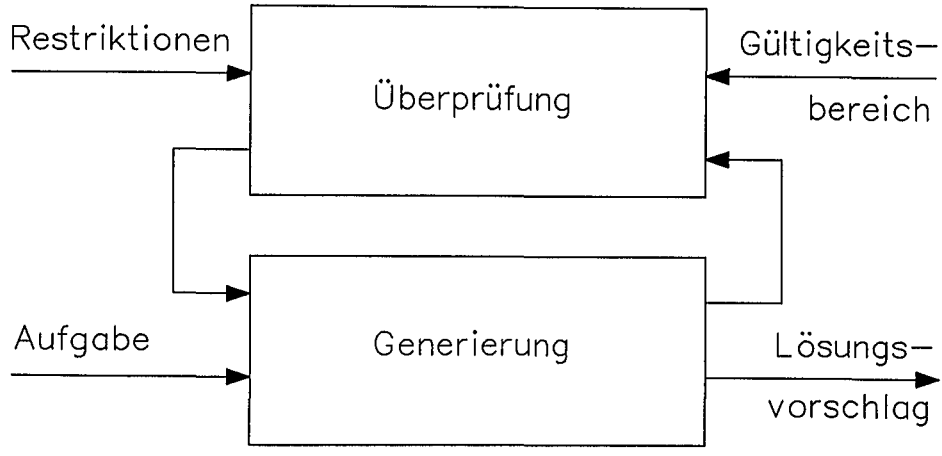


Bild 2.5: Schema zur Schaltungssynthese

Ein weiterer wichtiger Unterschied zur Softwarelösung ist durch die erheblich größere Vielfalt der Realisierungselemente gegeben. Wohingegen die strukturierte Programmierung nur wenige Grundelemente zum Aufbau von Modulen verwendet (z. B.: Anweisung; Auswahl; Iteration; Verkettung), ist bei der Hardwarelösung die Zahl der vom Anwender unmittelbar verwendeten Makrozellen vielfältig. Die Bilder 2.6 und 2.7 zeigen die Phasen der Software-Realisierung und die der Hardware-Realisierung. Bei der Softwaremethode erzeugen Compiler aus der höheren Programmiersprache selbständig einen effektiven, fehlerfreien, ablauffähigen Maschinencode. Das Ergebnis der Hardwareentwicklung hingegen ist ein dreidimensionales komplexes Schaltungsgebilde mit physikalischen Eigenschaften. Durch Steuerung von Ladungsflüssen und Potentialen werden logische Abläufe nachgebildet. Die Zugriffsmöglichkeiten zu Testsignalen sind stark eingeschränkt. In jeder Entwurfsphase muß die Testmethodik bedacht und berücksichtigt werden (siehe Bild 2.7). Die beiden Entwicklungswege sind funktional gleich, real aber gänzlich unterschiedlich.

Nach Analyse und funktionellem Entwurf wird ein komplexes System hierarchisch gegliedert und entsprechend einem Top-Down-Entwurf konzipiert. Einzelne Module sind miteinander verbunden und kommunizieren miteinander. Auf der obersten Hierarchiestufe bestehen die Module beispielsweise u.a. aus Prozessoren, Speicher, Controller. Verbunden werden die Module mit Datenbussen. Ein wichtiger Punkt ist die Festlegung der Schnittstellen.

Das hierarchische Konzept besteht nun darin, daß Module in Submodule soweit aufgeteilt werden, bis ein Submodul einfach genug ist, um in Hardware umgesetzt werden zu können.

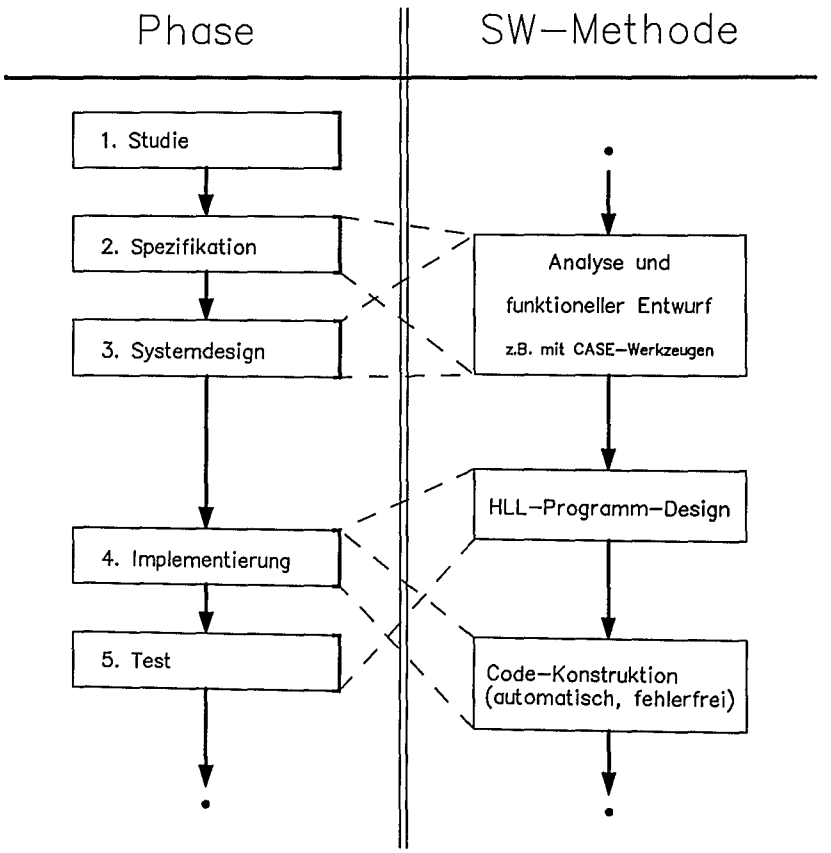


Bild 2.6: Systementwurf mit Software–Implementierung

Für die Analyse und für den funktionellen Entwurf in der Spezifikationsphase bzw. in der Systementwurfsphase werden heute u.a. CASE–Werkzeuge eingesetzt (CASE: Computer–Aided–Software–Engineering). Das Programm selbst wird in einer Hochsprache (HLL: High–Level–Language) entwickelt und implementiert.

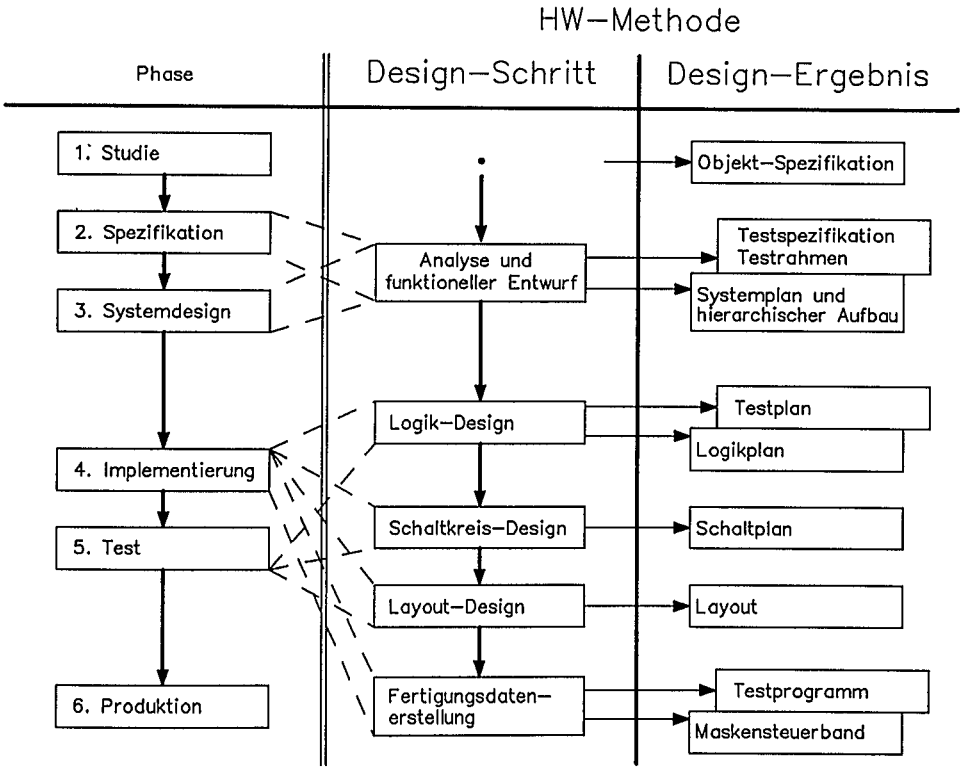


Bild 2.7: Systementwurf mit Hardware–Implementierung

Werkzeuge für den Systementwurf mit Hardware–Implementierung sind heute noch weitgehend im Forschungsstadium, z.B. das System DEBYS (DEBYS: Design_by_Specification /11/) mit dem Hardware–Generatorsystem GENSYS. Ein wichtiger Teil von den letztgenannten Entwurfshilfsmitteln ist ein Spezifikationsgenerator unter Einsatz einer Wissensbasis über die physikalische Realisierung, die zugrundeliegenden Restriktionen und dem Gültigkeitsbereich der mit Schaltungen realisierten Teilfunktionen.

Jede festgelegte Funktionseinheit muß vollständig und konsistent spezifiziert werden. Desweiteren müssen die Schnittstellen eindeutig beschrieben und festgelegt werden. Neben der Konsistenz und Vollständigkeit ist die Ausführbarkeit und Überprüfbarkeit einer Spezifikation ein wichtiges Kriterium. Bild 2.8 zeigt beispielhaft die Eingabemasken eines Spezifikationsgenerators für die einfache Grundfunktion "Zähler/Teiler".

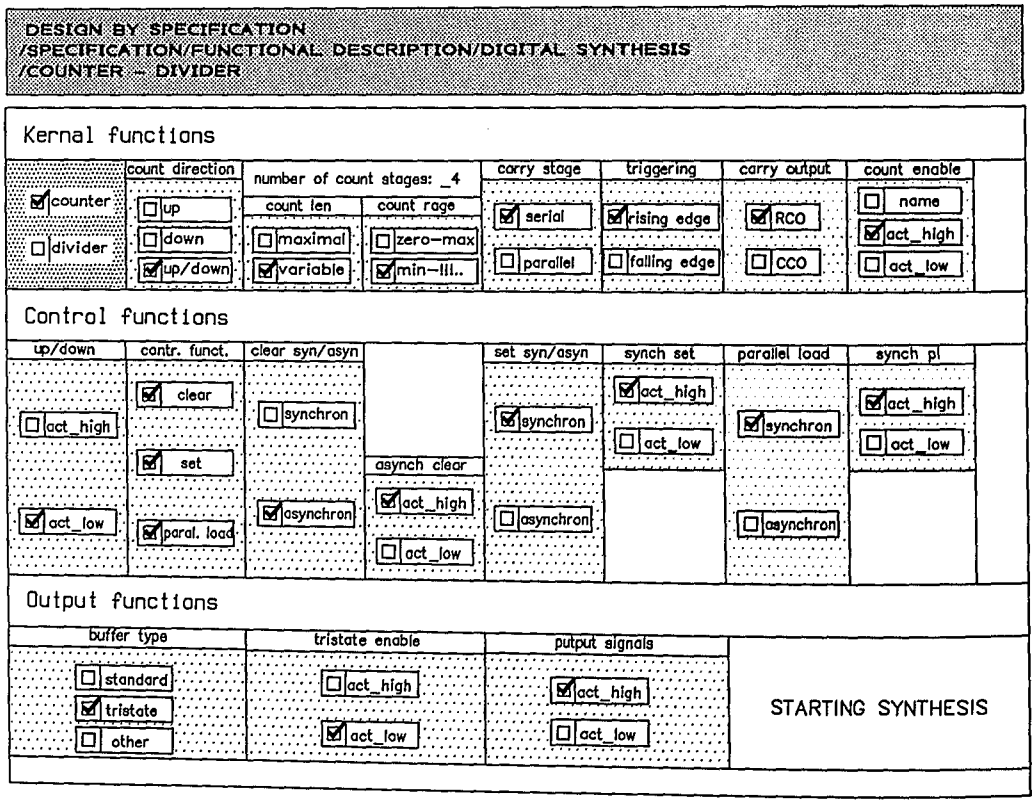


Bild 2.8: Beispiel einer Eingabemaske eines Spezifikationsgenerators

In Bild 2.9 ist die rechnergestützte Spezifikation eines hierarchischen Entwurfs skizziert. Dabei ergeben sich folgende Problemstellungen:

* Erstellung eines semantischen Modells einer Spezifikation;
* Darstellung des Modells mit geeigneten Datenstrukturen;
* Entwicklung von Mechanismen zur simulationsunabhängigen Konsistenzprüfung.

Im einzelnen muß eine Spezifikation mindestens folgende Angaben enthalten:

* Funktionale Spezifikation
* Schnittstellenspezifikation
* Beschreibung der Testbedingungen
* Komponentenbeschreibung
* Entwurfsmanagement

Zwischen den Spezifikationsparametern besteht eine Vielzahl von Relationen, die durch Regeln beschrieben werden können.

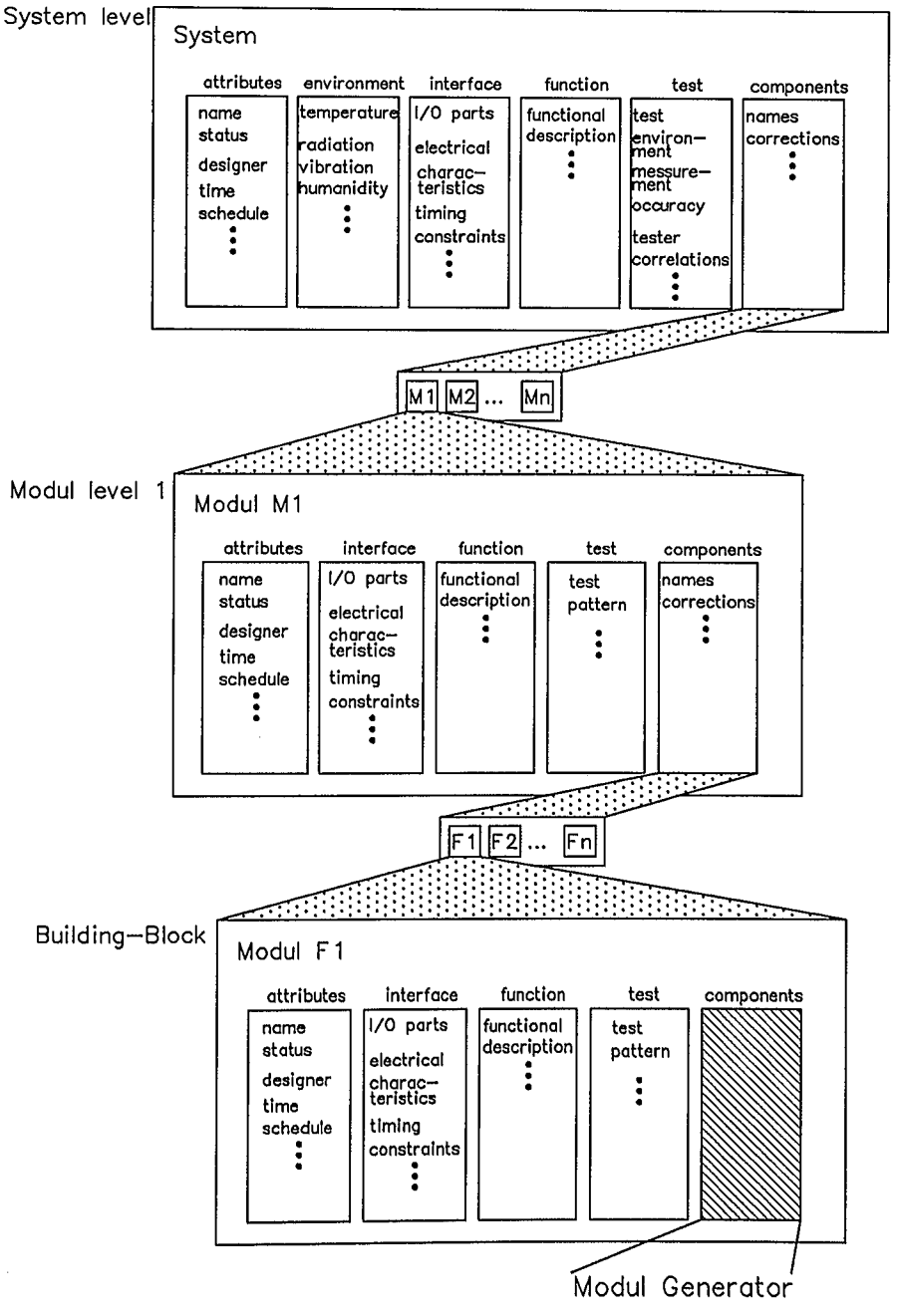


Bild 2.9: Zur Spezifikation in einem hierarchischen Entwurf

2.2.3 Einige Grundsätze zum Systementwurf

Die Zergliederung in Module soll so vorgenommen werden, daß jedem Modul eine bestimmte Funktion zugeordnet werden kann.

Auf der untersten Architekturebene sind die Module sogenannte Building-Blocks, die eine bestimmte abgegrenzte Funktion erfüllen und die für sich testbar sein sollten. Ein Building-Block sollte nicht mehr als ca. 2000 bis 3000 Gatterfunktionen beinhalten. Ein komplexes Design besteht demnach aus bis zu 50 Building-Blocks. Blöcke mit regulären Strukturen bilden dabei eine Ausnahme.

Aus Gründen der Testbarkeit sollten die Building-Blocks keine zu große sequentielle Tiefe aufweisen.

Ähnlich wie in der Software ist auch bei der Hardware das Prinzip der "Lokalität" für die Korrektheitsüberprüfung eines Moduls außerordenlich wichtig. Zur Verwirklichung der Lokalität müssen die nach außen gehenden Schnittstellen eines Moduls möglichst einfach gehalten werden. Damit bleibt ein Maximum an Information innerhalb eines Moduls. Globale Verdrahtungen sind möglichst über Busleitungen zu führen. Busleitungen können jedoch bei enger, paralleler Leiterführung bei höheren Signalverarbeitungsgeschwindigkeiten zu Problemen der Signalüberkopplung führen.

Vorteilhaft sind "Bit-Slice"-Strukturen. Nach Optimierung einer Grundfunktion zur Verarbeitung von z.B. einer 1-Bit-Funktion erfolgt der automatische Aufbau eines regulären Blocks zur Verarbeitung einer n-Bit-Funktion. Logische Funktionen, wie z.B. eine ALU können als 1-Bit-Funktionseinheit entworfen werden, die dann n-mal aneinandergereiht einen n-Bit-Funktionsblock ergibt.

2.3 Die Entwurfsebenen und deren Darstellungsarten

Gajski und Kuhn führten 1983 mit ihrem Y-Diagramm ein Modell für die Entwurfsdarstellung ein (Bild 2.10) /12/. Die Entwurfsebenen sind dort als konzentrische Kreise visualisiert, die man folgendermaßen einteilt:

* Architekturebene;
* Algorithmische Ebene;
* Funktionale Ebene;
* Schaltkreisebene.

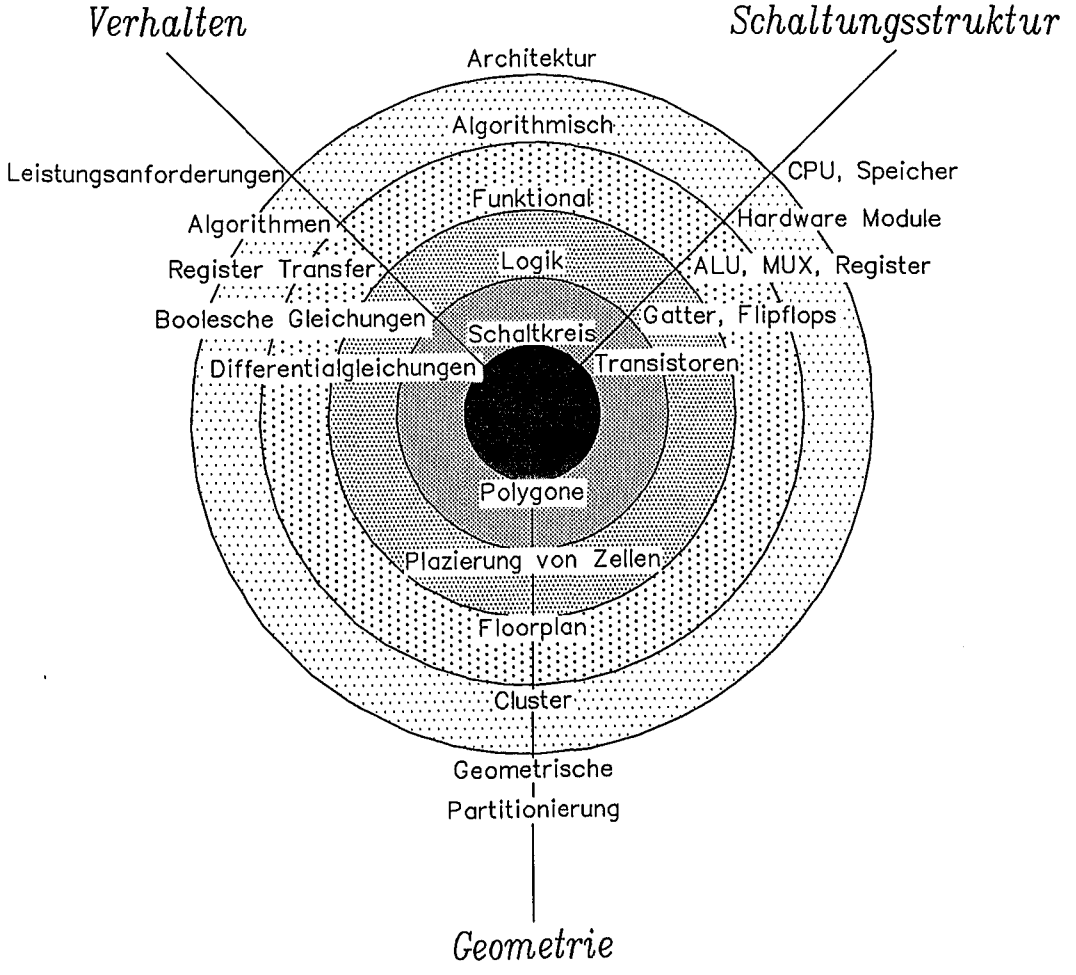
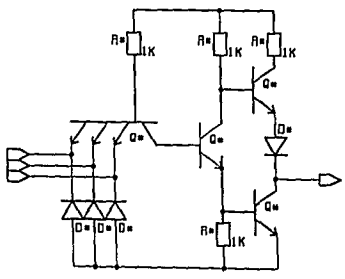


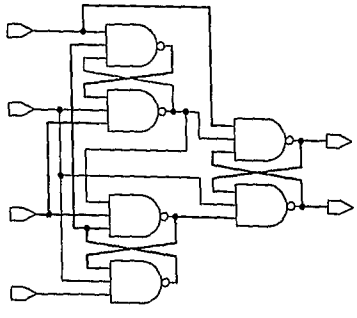
Bild 2.10: Zu den Entwurfsdarstellungsarten

Jede der Entwurfsebenen läßt sich darstellen im:
* Verhaltensbereich;
* Schaltungsstrukturbereich;
* Geometriebereich.

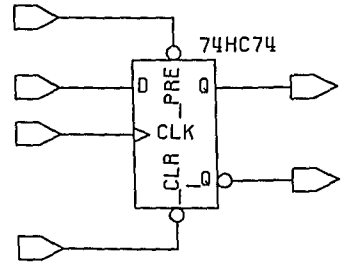
Verhaltensbereich: Der Verhaltensbereich beinhaltet Aussagen über die Funktion, die zugrundeliegenden Algorithmen, das logische Verhalten, das Zeitverhalten, die Zuverlässigkeit u. a. Zur Beschreibung der unterschiedlichen Aspekte werden verschiedene Beschreibungsformen verwendet. Das Verhalten einer Schaltung wird zunächst in natürlicher Sprache spezifiziert bzw. formuliert. Eine Verfeinerung der Verhaltensbeschreibung erfolgt durch Datenblätter, Ablaufdiagramme, Zeitdiagramme u. a. Die Werkzeuge zu den oberen Ebenen der Entwurfsdarstellung im Verhaltensbereich bestehen aus Sprachen zur Spezifikation des Verhaltensbereichs bzw. aus Compilern für prozedurale, applikative oder auch prädikatenlogische Programmiersprachen. In den unteren Ebenen wird das Verhalten durch Simulatoren und Analyseprogramme ermittelt. Bei der Modellbildung muß der Detaillierungsgrad dem betrachteten Abstraktionsgrad angepaßt sein. Bild 2.11 verdeutlicht den unterschiedlichen Detaillierungsgrad bei der Modellierung eines einfachen D–Flipflops.



a) Nand–Gatter auf Transistorebene



b) D–Flipflop auf Gatterebene



c) D–Flipflop als Makrozelle: Boolesche Gleichung oder Funktionstabelle

```
# If CLKRISE then
#    If PRE=1 and CLR=1 THE
#       If D=0 then
#          Q=0, QBAR=1
#       else
#          Q=1, QBAR=0
#    else ...
# else ...
```

d) Prozedurales Verhaltensmodell eines D–Flipflops

Bild 2.11: Modelle für unterschiedliche Detaillierungsgrade von Logikfunktionen

Schaltungsstrukturbereich: Der Schaltungsstrukturbereich besteht aus Komponenten und die sie verbindenden Netze. Komponenten sind in der

* Architekturebene: Prozessoren, Speicher, Controller, Datenbusse u. a.
* Registertransferebene: Register, Multiplexer, Codierer, Decodierer, ALU u. a.
* Logikebene: Flipflops, Gatter, Transmission-Gates u. a.
* Schaltkreisebene: Transistoren, Widerstände, Kondensatoren, Leitungen u. a.

Die Beschreibung im Schaltungsstrukturbereich kann verbal durch Definition von Netzlisten, grafisch durch Block- oder Registertransferdarstellungen, Logik-Schaltpläne, Transistor- oder "Stick"-Diagramme erfolgen. "Stick"-Diagramme sind symbolische Layouts auf Transistorebene; sie enthalten u.a. auch Informationen über die geometrische Lage der Transistoren zueinander.

Geometriebereich: Der Geometriebereich beschreibt die physikalische Implementierung einer Schaltung vektoriell durch Polygonzüge oder durch absolute Positionen. In höheren Ebenen wird durch den "Floorplan" die geometrische Aufteilung eines Chips festgelegt. Durch ein geeignetes "Floorplanning" kann eine gute Anpassung der Schaltungskomponenten erzielt werden. Als Schnittstelle zum Hersteller sind Standard- Geometriebeschreibungssprachen bzw. Geometriebeschreibungsformate (z. B. GDSII; CIF) eingeführt. Die Werkzeuge im Geometriebereich sind Layout-Editoren, Plazierungs- und Routing-Werkzeuge, "Design-Rule-Checker", Schaltungsextraktoren, Kompaktoren u. a.

Anzustreben ist eine isomorphe (von gleicher Gestalt) Darstellung der Blockfunktionen im Verhaltensbereich, Schaltungsstrukturbereich und Geometriebereich. Oftmals lassen sich gewählte hierarchische Dekompositionen nicht in andere Bereiche übertragen. Beispielsweise kann eine ALU mit Operandenregistern im Schaltungsstrukturbereich als Registertransfer-Struktur und im Geometriebereich als "Bit-Slice"-Struktur (z. B. 16-Bit-Scheiben) dargestellt werden.

2.4 Logiksynthese von endlichen Zustandsautomaten

Die Schaltungssynthese ist allgemein in Teilbereichen möglich. Im Analogbereich gibt es Verfahren zur Schaltungssynthese für bestimmte Funktionsschaltungen, wie z.B. Filterschaltungen. Die in diesem Abschnitt behandelte Logiksynthese beschränkt sich auf die Synthese von Zustandsautomaten für den Entwurf von Ablaufsteuerungen (ASM: Algorithmic State Machine). Das zugrundeliegende Konzept des logischen Entwurfs von Zustandsautomaten zeigt Bild 2.12. Der endliche Zustandsautomat besteht aus einem Registerblock (z.B. D–; T–; RS– oder JK–Flipflops) und einem Block mit kombinatorischer Logik. Der Registerblock beinhaltet den Zustand der Schaltung. Dadurch wird ein einzelner Verarbeitungsschritt gekennzeichnet. Die kombinatorische Schaltung bildet aus dem aktuellen Zustand S^t und den Eingangssignalen **X** den gewünschten Folgezustand S^{t+1} und die vorgesehenen Ausgangssignale Y_1 (Mealy–Automat). Während die Ausgänge Y_1 sich innerhalb eines Zustandes ändern können, sind die Ausgänge Y_2 getaktet und frei von möglichen Fehlimpulsen (Moore–Automat).

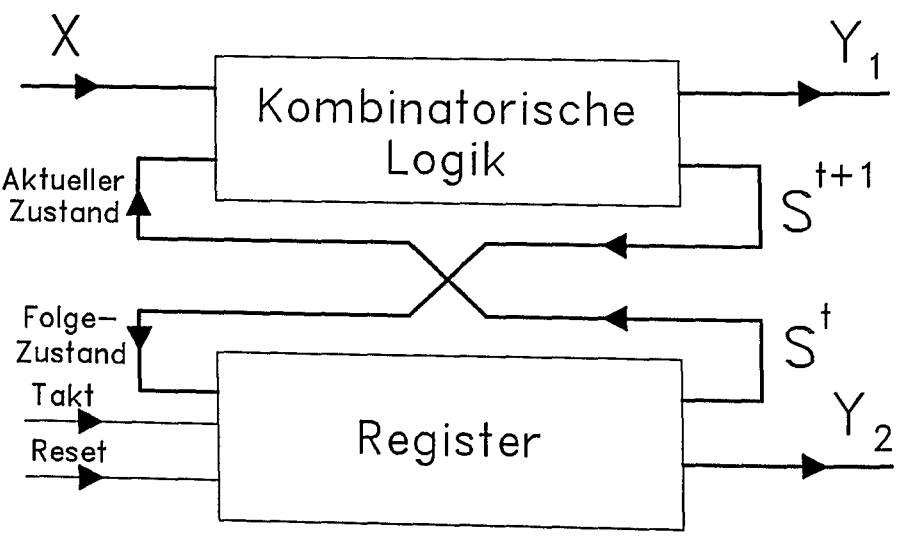


Bild 2.12: Grundkonzept eines endlichen Zustandsautomaten

Der logische Entwurf einer Ablaufsteuerung ist Teil des Systementwurfs. Im Systementwurf werden die Funktionsblöcke festgelegt. Funktionsblöcke können z.B. Registerblöcke sein. Der Datenfluß zwischen den Funktionsblöcken wird im allgemeinen durch eine Ablaufsteuerung gesteuert.

Bild 2.13 zeigt den logischen Entwurf z.B. einer Ablaufsteuerung als Teil des Systementwurfs. Die aus dem logischen Entwurf gewonnenen Booleschen Gleichungen müssen dann in einen topologischen Entwurf umgesetzt werden. Anschließend kann die Implementierung in einer bestimmten Technologie erfolgen.

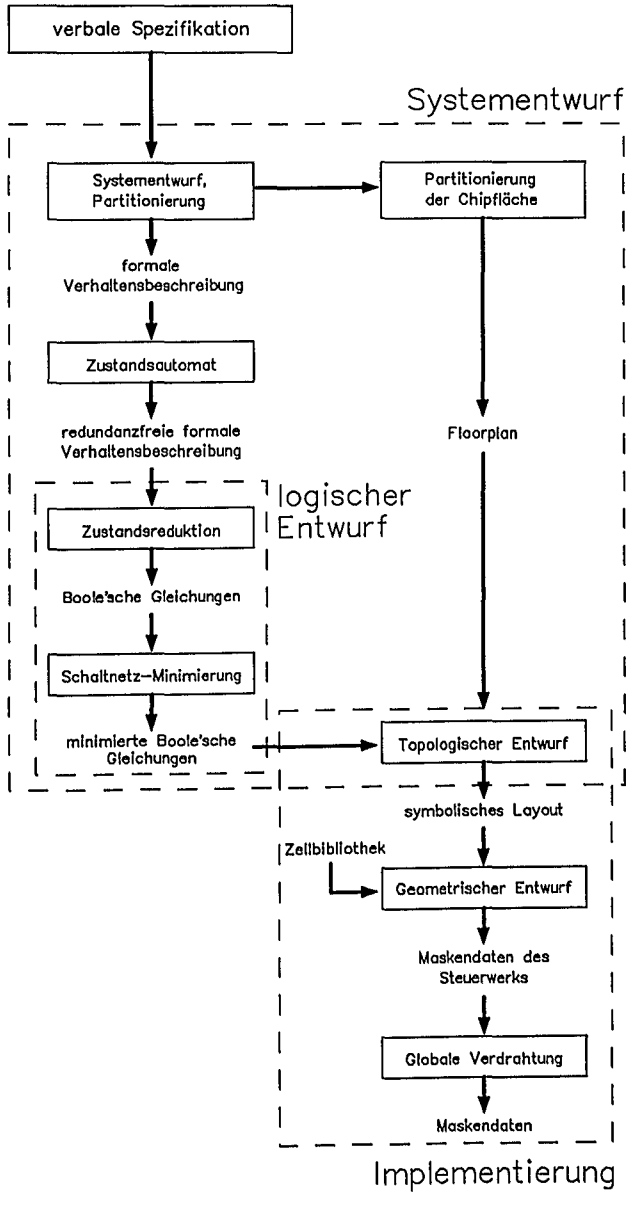


Bild 2.13: Gliederung des Entwurfsablaufs von Steuerwerken

Grundsätzlich läßt sich eine Ablaufsteuerung durch einen Zustandsautomaten realisieren. Bei manueller Vorgehensweise ist zunächst für die Ablaufsteuerung ein Ablaufdiagramm zu entwerfen. Sodann ist aus dem Ablaufdiagramm ein Zustandsübergangsdiagramm zu ermitteln. Bild 2.14 zeigt ein Ablaufdiagramm; es besteht im allgemeinen aus Ausführungsanweisungen und Sprunganweisungen. Jedem Ausführungsschritt wird ein Zustand S_i zugeordnet. Die Zustände S_i werden in geeigneter Weise durch Registerzustände codiert. Bei 3 Register–Flipflops ist es demnach möglich 8 Ausführungsschritte zu codieren. Zur Veranschaulichung soll ein einfaches Beispiel in Form eines digitalen Impulsvergleichers gewählt werden.

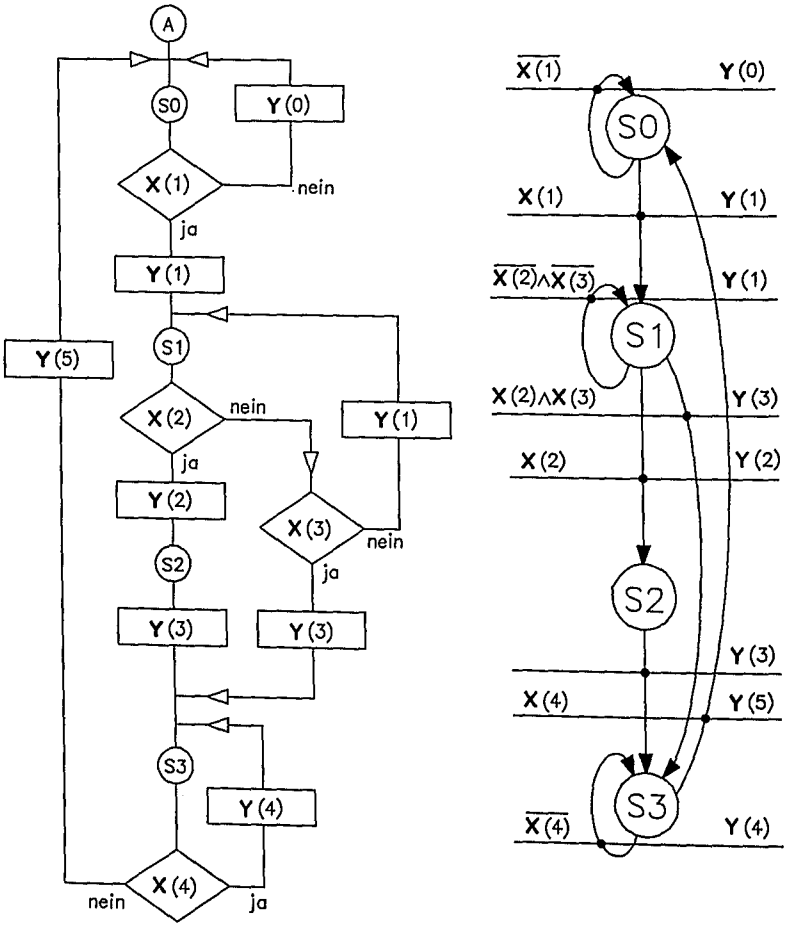


Bild 2.14: Allgemeines Ablaufdiagramm und zugehöriges Zustandsübergangs-diagramm; X(i): Eingangssignalbedingung Y(i): Ausgangssignalbedingung

Funktionsspezifikation des ersten Beispiels: Zwei Impulse X_1 und X_2 sind hinsichtlich der Phasenlage zueinander (Bild 2.15a) zu vergleichen. X_1 muß vor X_2 erscheinen und verschwinden, noch während X_2 vorhanden ist. Bei einer Abweichung ist ein Fehlersignal Y auszugeben. Nach einer Fehlererkennung soll das Schaltwerk im Fehlerzustand verharren. Ein RESET–Signal stellt den Ausgangszustand wieder her.

Bild 2.15b zeigt das Ablaufdiagramm und Bild 2.15c das Zustandsübergangs–diagramm für das gewählte Beispiel.

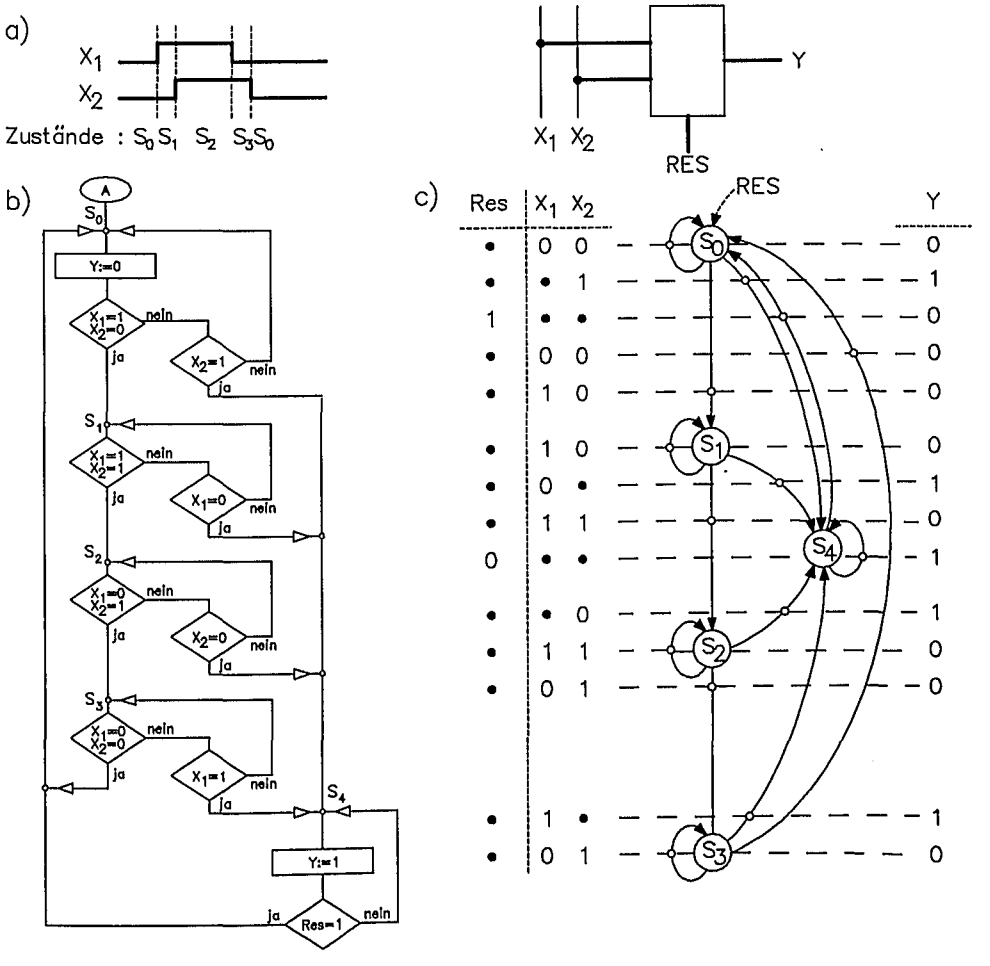
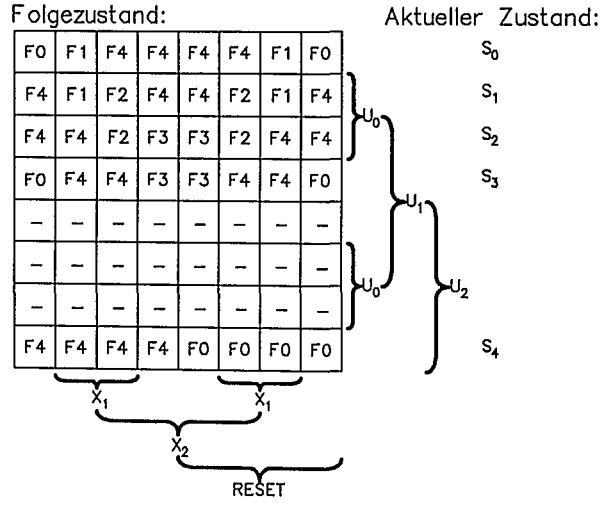


Bild 2.15: Logischer Entwurf von Beispiel 1: a) Zur Aufgabenstellung, b) Ablaufdiagramm, c) Zustandsübergangsdiagramm

Aus dem Zustandsübergangsdiagramm kann bei gegebener Eingangssignalbelegung und gegebenem Zustand der Folgezustand ermittelt werden. Möglich ist eine Darstellung in Form einer KV-Tafel (Bild 2.16a) oder in textueller Form (Bild 2.16b).

a)



b)

		RES	X1	X2		Y
State 0	S0:	–	0	0	→	F0: 0
	S0:	–	1	0	→	F1: 0
	S0:	–	–	1	→	F4: 1
State 1	S1:	–	1	0	→	F1: 0
	S1:	–	1	1	→	F2: 0
	S1:	–	0	–	→	F4: 1
State 2	S2:	–	1	1	→	F2: 0
	S2:	–	0	1	→	F3: 0
	S2:	–	–	0	→	F4: 1
State 3	S3:	–	0	1	→	F3: 0
	S3:	–	0	0	→	F0: 0
	S3:	–	1	–	→	F4: 1
State 4	S4:	0	–	–	→	F4: 0
	S4:	1	–	–	→	F0: 1

Bild 2.16: Verschiedene Darstellungen der Zustandsübergangstabelle
a) KV-Tafel-Darstellung b) Textuelle Darstellung

Wie nun die Synthese der Ablaufsteuerung durchgeführt wird, ist im folgenden dargestellt. Zunächst müssen die Zustände codiert werden (Bild 2.17a). Bild 2.17b zeigt die KV-Tafeldarstellung unter Berücksichtigung der gewählten Zustandscodierung. Die Art der Zustandscodierung ist frei wählbar, sie bestimmt mit den Schaltungsaufwand. Oftmals wird ein Gray-Code gewählt, da sich dabei von Zustand zu Zustand nur ein Bit verändert. Im allgemeinen ist dabei auch der Schaltungsaufwand geringer. Als nächstes erfolgt die Auswahl der Flipflop-Art für das Zustandsregister. In der Regel werden D-Flipflops oder JK-Flipflops gewählt. Bei JK-Flipflops ist eine hohe Funktionssicherheit durch synchrone Master-Slave-Flipflops mit asynchronem Reset gegeben.

a) Zustandscodierung

	U_0	U_1	U_2	
S_0	0	0	0	$\overline{U_0}\ \overline{U_1}$
S_1	1	0	0	$U_0\ \overline{U_1}$
S_2	1	1	0	$U_0\ U_1$
S_3	0	1	0	$\overline{U_0}\ U_1$
S_4	0	0	1	U_2

b) Zustandsübergangstabelle als KV-Tafel

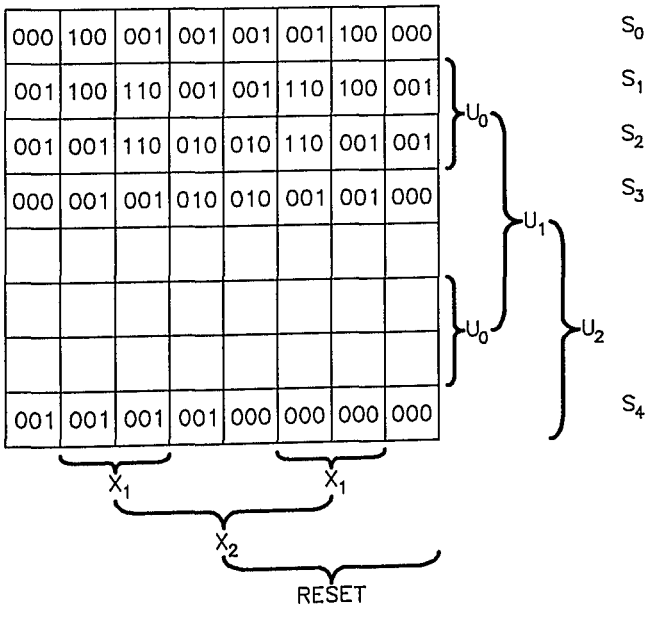
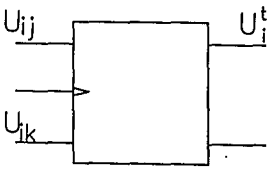


Bild 2.17: Zustandscodierung von Beispiel 1

Für unser Beispiel wurden JK-Flipflops für das Zustandsregister gewählt. Bild 2.18a zeigt die Boolesche Gleichung für das JK-Flipflop. Daraus ergibt sich die logische Bedingung für den J- bzw. K-Eingang. In unserem Fall ist U_i^t gegeben; mit U_{ij} ; U_{ik} wird der gewünschte Folgezustand eingestellt. Bei $U_i^t = 0$ ist $U_{ij} = U_i^{t+1}$ und bei $U_i^t = 1$ ist $U_{ik} = \overline{U}_i^{t+1}$. Die konkrete Ausführung am Beispiel zeigt Bild 2.19. In Bild 2.18b sind als Ergebnis die Booleschen Gleichungen für die Zustandsregister-Flipflops zusammengefaßt.

a) JK-Flipflop



U_i^t	U_i^{t+1}	U_{ij}	U_{ik}
0	0	0	–
0	1	1	–
1	0	–	1
1	1	–	0

$$U_i^{t+1} = U_{ij}^t \, \overline{U}_i^t + \overline{U}_{ik}^t \, U_i^t$$

$$U_{ij} = U_i^{t+1}\Big|_{U_i=0}$$

$$U_{ik} = \overline{U}_i^{t+1}\Big|_{U_i=1}$$

b) Boolesche Gleichungen der Zustandsregister (realisiert mit JK-Master-Slave-Flipflops)

$$U_{0j} = \overline{U}_1 \cdot \overline{U}_2 \cdot X_1 \cdot \overline{X}_2$$
$$U_{0k} = X_2 \cdot \overline{X}_1 \vee \overline{X}_2 \cdot U_1 \vee \overline{X}_1 \cdot \overline{X}_2$$

$$U_{1j} = X_1 \cdot X_2 \cdot U_0$$
$$U_{1k} = \overline{X}_2 \vee X_1 \cdot \overline{U}_0$$

$$U_{2j} = X_2 \cdot \overline{U}_0 \cdot \overline{U}_1 \vee \overline{X}_1 \cdot U_0 \cdot \overline{U}_1 \vee \overline{X}_2 \cdot U_0 \cdot U_1 \vee X_1 \cdot \overline{U}_0 \cdot U_1$$
$$U_{2k} = \text{RESET}$$

Bild 2.18: a) Ausgewählte Flipflop-Art b) Boolesche Gleichungen der J- und K-Eingänge

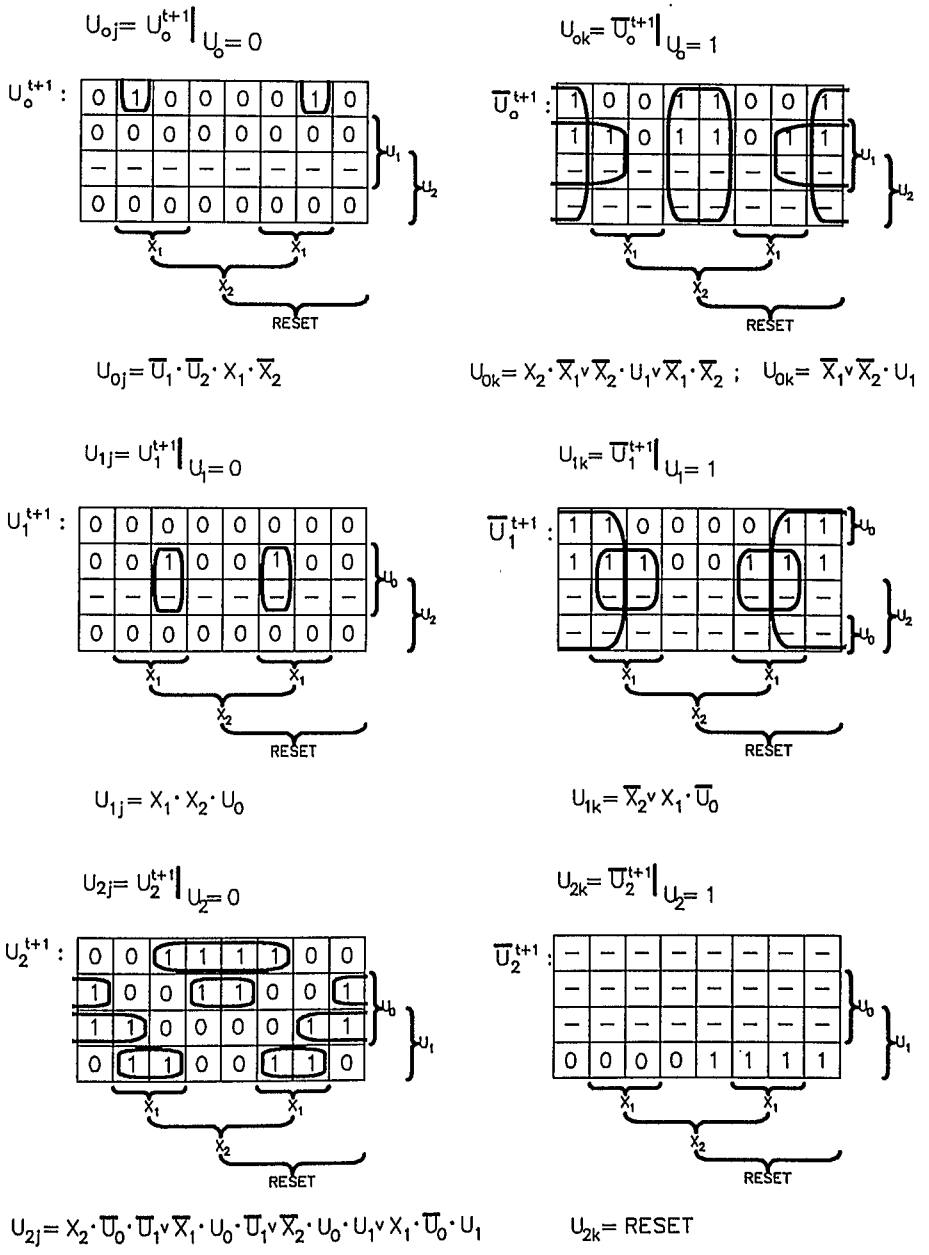


Bild 2.19: Zur Ermittlung der Booleschen Gleichungen für die J– und K–Eingänge des Zustandsregisters

Die minimierten Booleschen Gleichungen für die Registereingänge und für die Ausgangssignale werden nun durch entsprechende Gatterbeschaltungen realisiert; es wird eine Schaltungstopologie erzeugt. Bild 2.20a zeigt die Schaltung als Ergebnis der Logiksynthese. In Bild 2.20b wird die Schaltung durch Logiksimulation getestet.

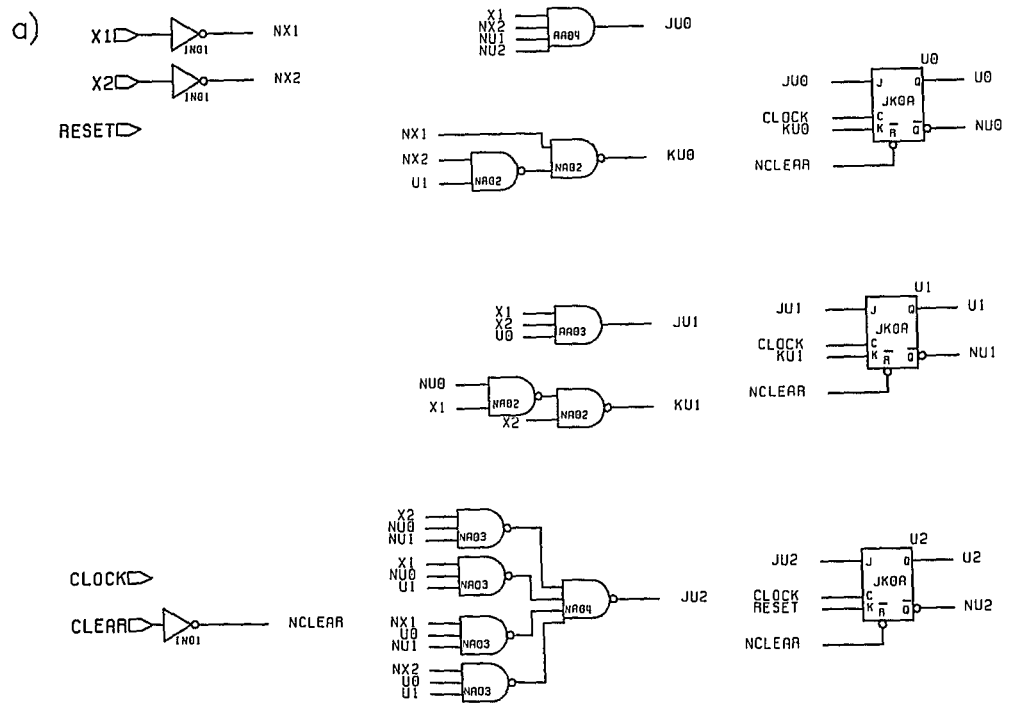


Bild 2.20: Beispiel 1: Schaltung a) und Ergebnis b) der Logiksimulation

Funktionsspezifikation des zweiten Beispiels: Es ist eine Ablaufsteuerung zur Erzeugung taktsynchroner Steuerimpulse Y zu entwerfen. Die Länge der Steuerimpulse Y soll durch den Eingang X_1 und X_2 eingestellt werden. Ausgehend von der Taktfrequenz 2 MHz werden die exakten Zeiten über einen Binärzähler Z erzeugt. Der Startzeitpunkt wird durch das Signal START festgelegt. Unmittelbar nach START soll der Zähler Z und auch die Ablaufsteuerung mit dem Resetsignal R in einen definierten Ausgangszustand gebracht werden. Der Steuerimpuls Y soll bei t = 0 beginnen und bis zum angegebenen Zeitpunkt andauern.

In dem gewählten Beispiel ist ein Steuersignal **Y** bestimmter Länge zu erzeugen. Die Länge des Steuersignals ist abhängig von **X**. Der Beginn der Zeitbasis ist in Bild 2.21 dargestellt.

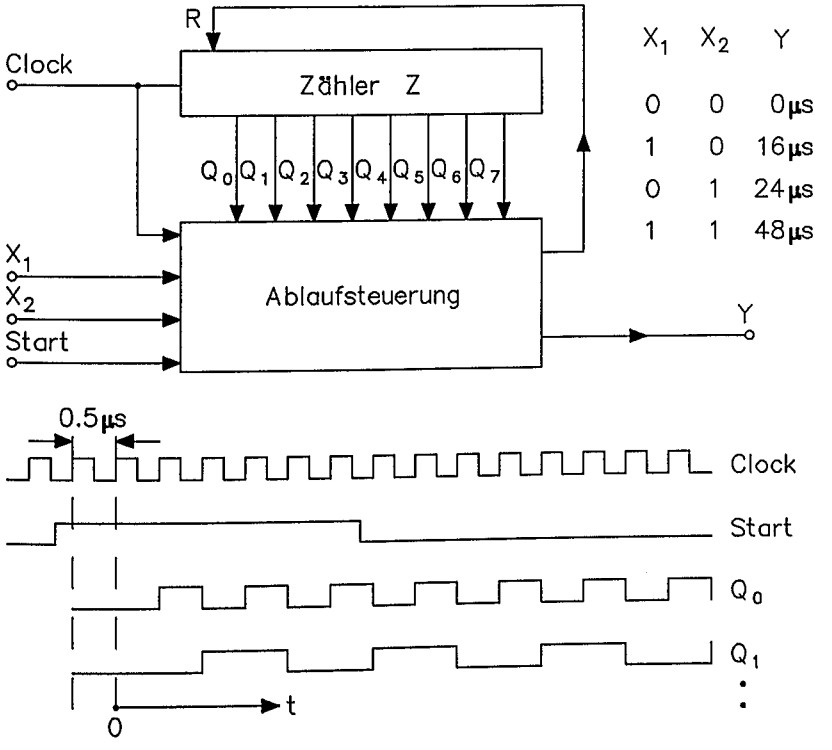


Bild 2.21: Blockschaltbild und Signalzustände zum zweiten Beispiel

Die Ablaufsteuerung ist durch drei Zustände gekennzeichnet: den Wartezustand S_1 , den Vorbereitungszustand S_2 und den Ausführungszustand S_3 . Bild 2.22 zeigt das zugehörige Zustandsübergangsdiagramm. Die Länge des Steuersignals Y hängt ab von den Ausgangssignalen des Zählers Z.

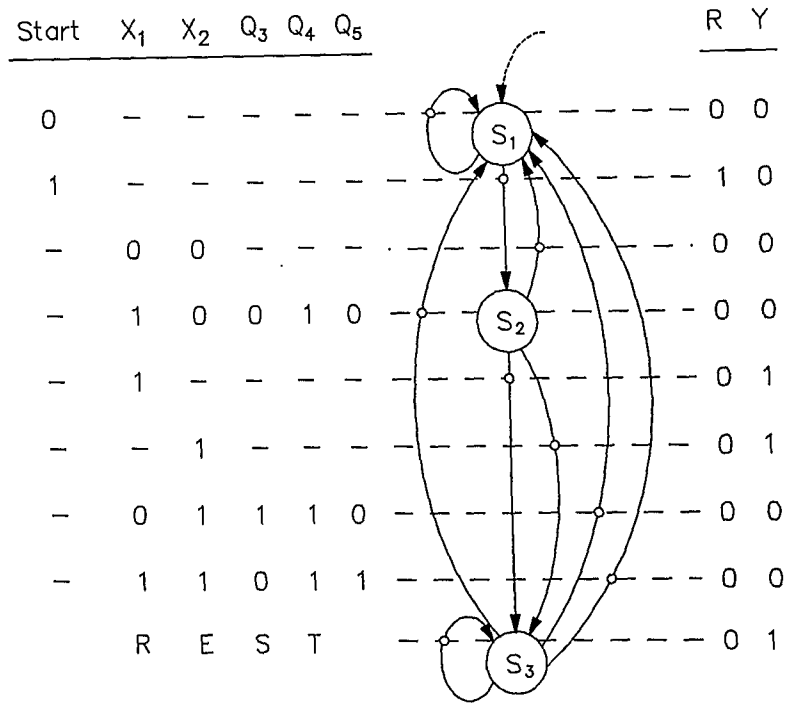


Bild 2.22: Zustandsübergangsdiagramm

Mit Logiksynthesewerkzeugen kann aus dem Zustandsübergangsdiagramm die dafür erforderliche Schaltung synthetisiert werden. Bild 2.22 zeigt den Zustands-übergangsgrafen. Möglich ist auch eine textuelle Darstellung, wie für das erste Beispiel angegeben und in Bild 2.16b dargestellt.

Funktionsspezifikation des dritten Beispiels: Für die in Bild 2.23 skizzierte Multiplizierschaltung zweier 4stelliger Dualzahlen ist eine Ablaufsteuerung zu entwerfen. Für die Ablaufsteuerung steht ein kontinuierlicher Takt *CLOCK* zur Verfügung. Die Multiplikation soll durch ein externes Startsignal *START* ausgelöst werden. Die zu multiplizierenden Operanden sind über das Operandenregister *OR* einzugeben. Durch das Übernahmesignal *NEXT* wird das Anliegen des zweiten Operanden gekennzeichnet. Das Ergebnis soll am Schluß der Multiplikation im *AC–MQ* Register stehen; die Bereitstellung des Ergebnisses wird durch das *READY*–Signal markiert. Das *RESET*–Signal bringt die Multiplizierschaltung in einen definierten Ausgangszustand.

Dieses Beispiel beinhaltet eine Logikschaltung zur Ablaufsteuerung einer Schaltungsstruktur auf Registertransferebene.

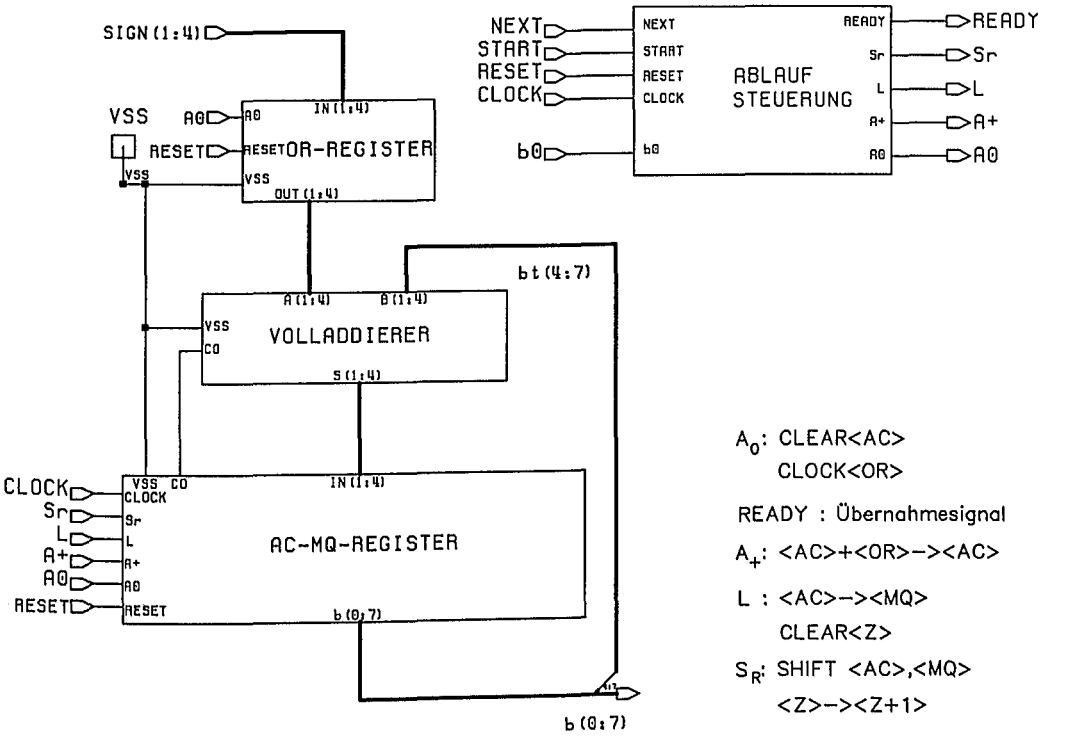


Bild 2.23: Blockschaltbild der Multiplizierschaltung

Ist der Multiplikator eine 4stellige Dualzahl, so ergeben sich 4 Teilprodukte. Diese Teilprodukte müssen je nach Wertigkeit verschoben werden. Das Ergebnis der Multiplikation erhält man z.B. durch Addition der Teilprodukte. Das Prinzip der binären Multiplikation zeigt Bild 2.24.

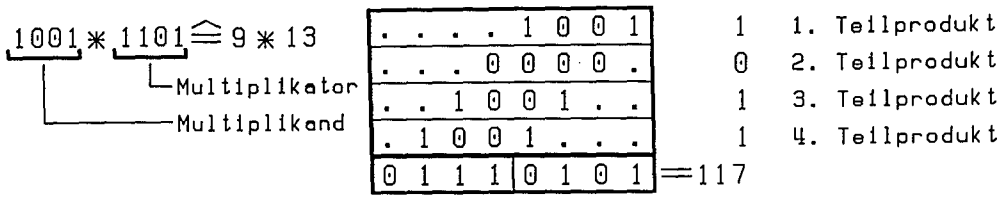


Bild 2.24: Beispiel: Prinzip einer binären Multiplikation

Jedes Teilprodukt ist gleich 0000 oder gleich dem Multiplikand 1001. Entsprechend der Wertigkeit ist die Teilsumme vor der neuen Addition des Teilproduktes um eine Stelle zu verschieben. Für die Multiplikation benötigt man neben einem Addierer ein Multiplikator-Register. Im Beispiel ist der Addierer ein Volladdierer, damit läßt sich eine Paralleladdition vornehmen. Die Multiplikation kann in Einzelschritte aufgelöst werden, wie nachstehendes Beispiel zeigt:

1 0 0 1			
0 0 0 0	1 1 0 **1**	⇒	$<MQ_0>=1$, 1. Stelle
1 0 0 1	1 1 0 1	⇒	Addition
0 1 0 0	1 1 1 **0**	⇒	Shift, $<MQ_0>=0$, 2. Stelle
0 0 1 0	0 1 1 **1**	⇒	Shift, $<MQ_0>=1$, 3. Stelle
1 0 1 1	0 1 1 1	⇒	Addition
0 1 0 1	1 0 1 **1**	⇒	Shift, $<MQ_0>=1$, 4. Stelle
1 1 1 0	1 0 1 1	⇒	Addition
0 1 1 1	0 1 0 1	⇒	Shift
0 1 1 1	**0 1 0 1**		**ERGEBNIS**

Die Einzelschritte werden durch Steuerausgangssignale (Steuersignale bzw. Steuer-Variable) veranlaßt, ausgelöst von den Steuereingangssignalen und Entscheidungsvariablen. In unserem Beipiel benötigen wir folgende Signale zur Steuerung des Ablaufs der Multiplikation:

Steuereingangssignale:

START=1	**START**
<NEXT>=1	**NEXT**

Entscheidungsvariable:

$<MQ_0>=1$	b_0
<Z>=4	**Z**

Steuerausgangssignale:

CLEAR<AC>; CLOCK<OR>	A_0
READY	**READY**
<AC>+<OR>→<AC>	A_+
<AC>→<MQ>; CLEAR<Z>	**L**
shift <AC>,<MQ>; <Z>→<Z+1>	S_R

Als nächstes wird ein logischer Ablaufplan (Bild 2.25a) erstellt. Jede Anweisung (zeitlich getrennt von einer anderen Anweisung), die durchgeführt werden muß, ist bei einem Zustandswechsel zu veranlassen. Daraus läßt sich ein Zustandsübergangs-diagramm zur Steuerung der Multiplikation ableiten (Bild 2.25b).

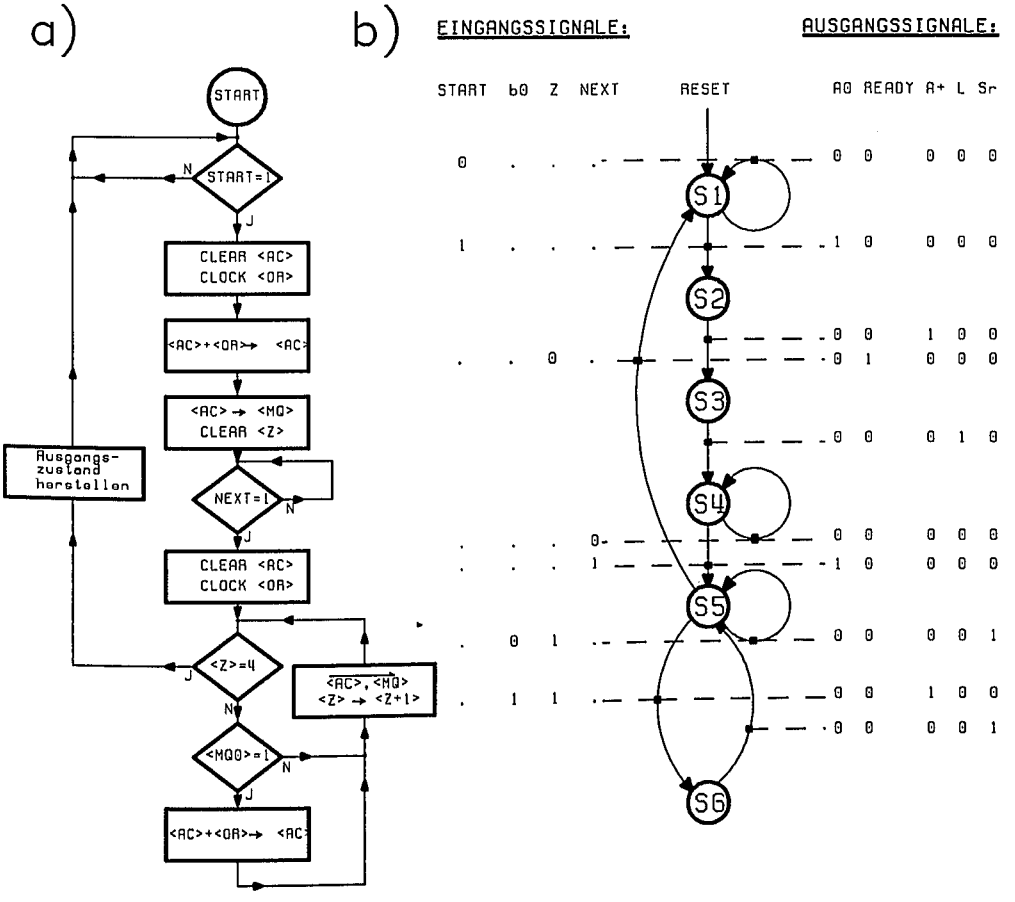


Bild 2.25: Ablaufsteuerung a) Ablaufdiagramm b) Zustandsübergangsdiagramm

Nach Durchführung einer automatischen Logiksynthese erhält man die Schaltungsstruktur der Ablaufsteuerung. In Bild 2.26 ist das Schaltbild des Zustandsregisters und das Schaltnetz der Ausgangssignale angegeben. Um Fehl-impulse zu vermeiden, sind einige Ausgangssignale getaktet. Bild 2.27 schließlich zeigt das Ergebnis der Logiksimulation der gesamten Multiplizierschaltung.

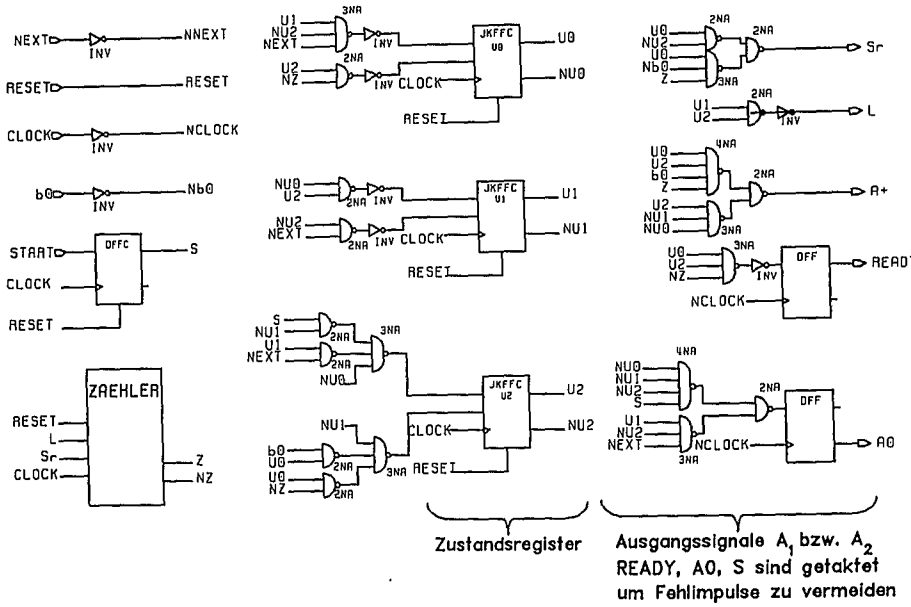


Bild 2.26: Schaltbild der realisierten Ablaufsteuerung

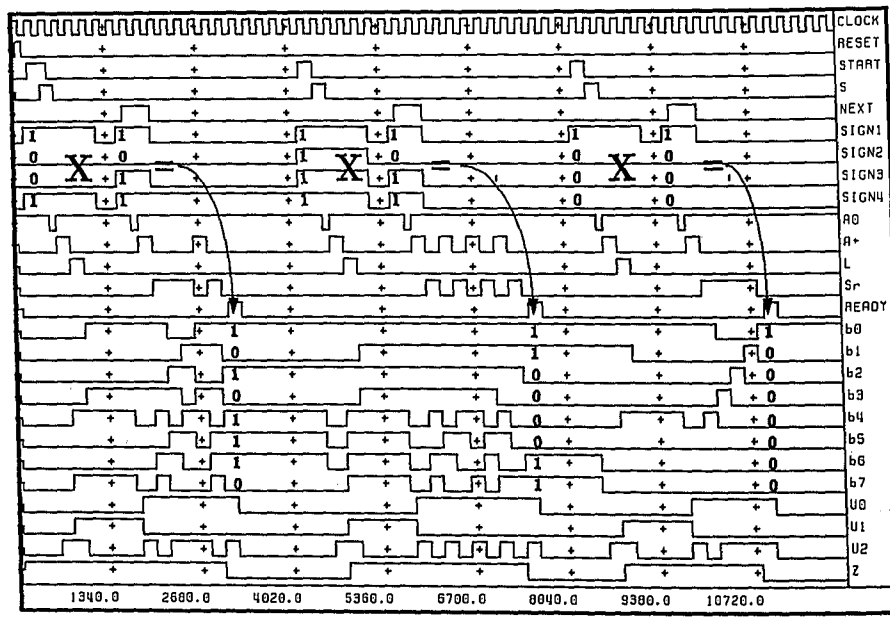


Bild 2.27: Ergebnis der Logiksimulation

Zum Abschluß dieses Abschnittes soll durch Verallgemeinerung der Problemstellung des letzten Beispiels übergeführt werden auf den nächsten Abschnitt. Bild 2.28 zeigt das Grundschema einer Registerstruktur zur Realisierung eines Algorithmus zur Signalverarbeitung von Daten, gesteuert durch eine Ablaufsteuerung. Die Registerstruktur empfängt Daten und liefert verarbeitete Daten. Der Ablauf der Verarbeitung wird gesteuert durch die Steuerausgangssignale, die wiederum werden veranlaßt durch die Steuereingangssignale und Entscheidungsvariable.

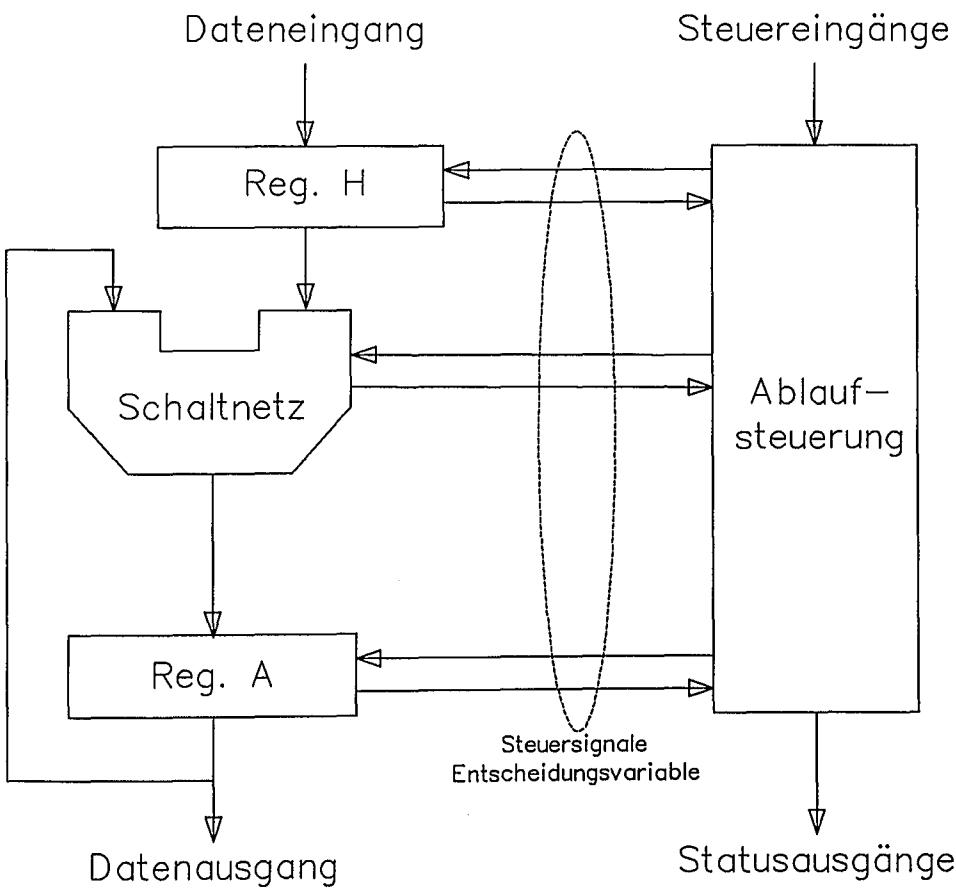


Bild 2.28: Registerstruktur mit Ablaufsteuerung zur Realisierung eines Algorithmus zur Signalverarbeitung von Daten

2.5 Höhere Abstraktionsebenen von digitalen Funktionen

2.5.1 Zur Registertransfer–Beschreibung

Für den Entwurf komplexer Elektroniksysteme ist eine höhere Abstraktionsebene erforderlich. Systeme mit 100.000 Gatterfunktionen und mehr können nicht auf Gatterebene sinnvoll entwickelt werden. Durch Einführung von Blockfunktionen und durch ein hierarchisches Konzept läßt sich auf einer höheren Abstraktionsebene die Funktionalität beschreiben. Die algorithmische Ebene ist die heute mit Entwurfswerkzeugen darstellbare höchste Beschreibungsebene. Auf einer höheren Beschreibungsebene gewinnt man an Übersicht, es lassen sich Realisierungsalternativen leichter verifizieren. Gleichzeitig wird die Menge der Daten für die Entwurfsdarstellung erheblich reduziert (Bild 2.29).

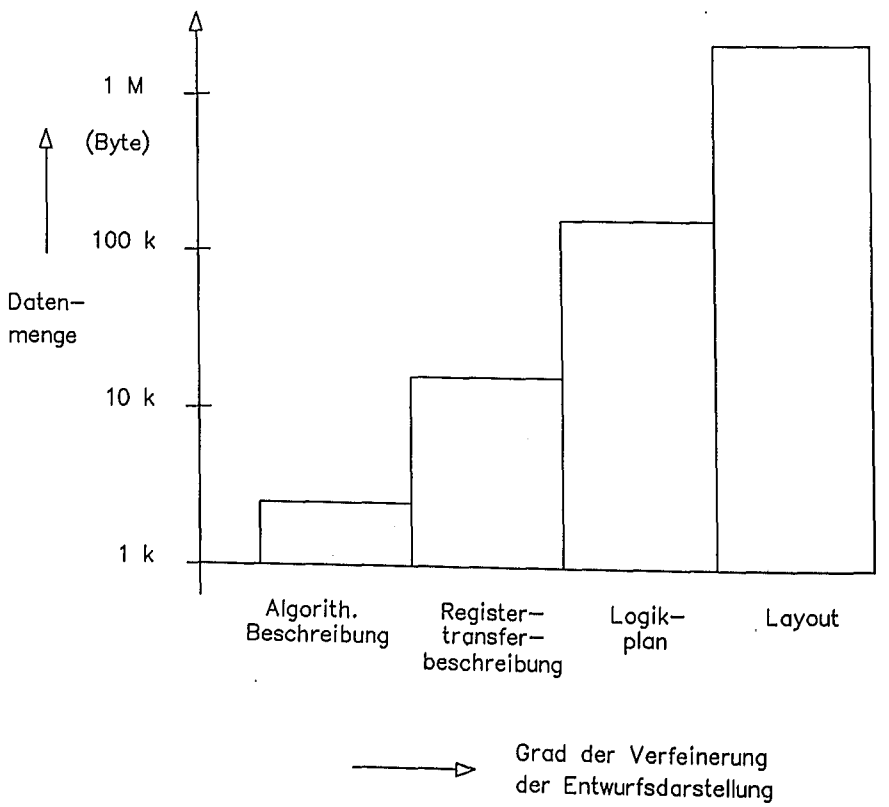


Bild 2.29: Beispiel für die erforderliche Datenmenge bei unterschiedlichen Entwurfsdarstellungsarten

Registertransferbeschreibung: In der Registertransferbeschreibung wird der Datenfluß zwischen Logikfunktionen dargestellt. Die Datenflußbeschreibung erfolgt z.B. auf Wortebene. Ähnlich wie in einer Programmiersprache werden Verknüpfungen zwischen Variablen (Datenwörtern) und Zuweisungen an Variable formuliert. Diese Zuweisungen werden auch Mikroanweisungen genannt. Zumeist sind bestimmten Variablen Register in einer späteren Implementierung zugeordnet. Die Registertransferbeschreibung ist im Prinzip die Steuerung des Datenflusses auf Busebene über Register und deren Verknüpfungssteuerung. Prinzipiell unterscheidet man folgende Formen einer Registertransferbeschreibung:

* A <--- B	Datenfluß: übertrage Inhalt von Register B nach Register A;
* T: A <--- B	Bedingter Datenfluß: übertrage Inhalt von Register B nach Register A, falls die Kontroll-Bedingung T wahr ist; dabei kann T ein beliebiger Boolescher Ausdruck sein;
* Tϕ: A <--- B	Synchroner bedingter Datenfluß: übertrage Inhalt von Register B nach Register A, falls die Kontrollbedingung T wahr ist und das Taktsignal (aktive Clock-Flanke) aktiv ist;
* T: A <--- B+C	bedingter Datenfluß mit Operation auf gespeicherten Operanden.

Verdeutlicht werden soll die Registertransferbeschreibung am Beispiel des Operationswerkes eines Prozessors. Damit kann allgemein ein bestimmter Algorithmus zur Verarbeitung von Daten realisiert werden.

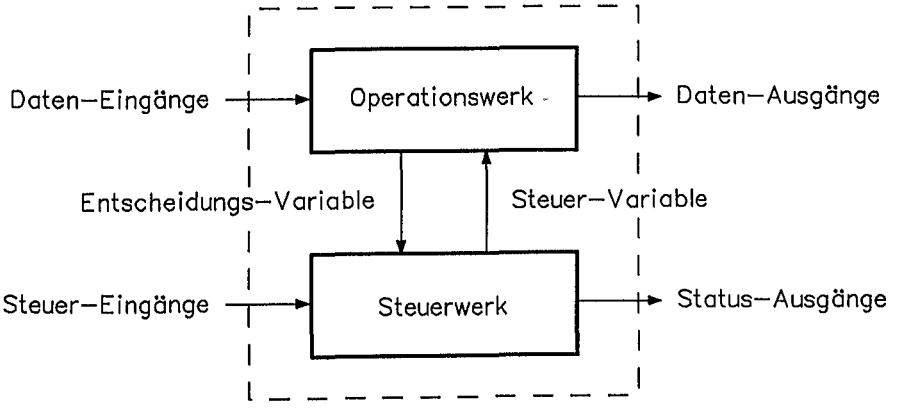


Bild 2.30: Beispiel: Operations- und Steuerwerk

In Bild 2.30 ist das Operationswerk als Funktionsblock dargestellt. Der Block empfängt Daten und gibt Daten wieder an nachgeordnete Funktionseinheiten ab. Der Datenfluß wird über Steuer–Variable, d.h. Steuersignale gesteuert. Die Steuersignale liefert das Steuerwerk; sie sind abhängig von Entscheidungsvariablen. Damit ist der Ablauf des Steuerwerkes abhängig von den Steuereingängen und von Entscheidungsvariablen des Operationswerkes. Das Operationswerk selbst besteht aus einer Registerstruktur mit Datenleitungen und Steuerleitungen. Der Datenfluß erfolgt über die Datenleitungen gesteuert durch die Steuersignale. Die Blockstruktur des Operationswerkes ist in Bild 2.31 dargestellt, mit dem Datenfluß zwischen den Registerblöcken. Das Schaltnetz SN ist eine arithmetisch/logische Einheit (ALU) zur Ausführung von Grundoperationen. Konkret wird der Datenfluß in Abhängigkeit des decodierten Befehls über Ablaufsteuerungen gesteuert.

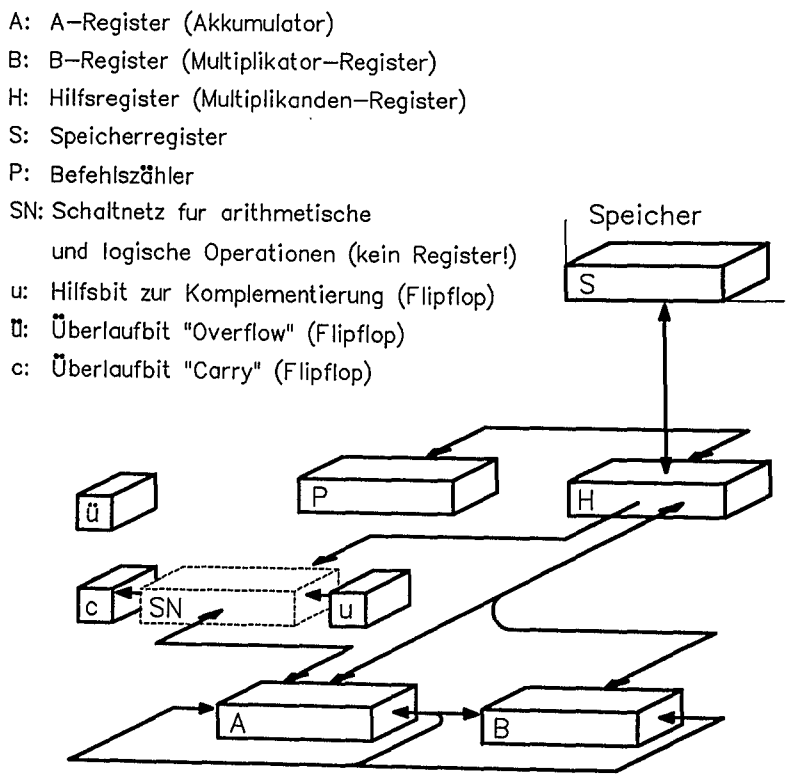


Bild 2.31: Blockschaltbild eines Operationswerkes mit Busleitungen für den Datenfluß zwischen den Registerblöcken

Bild 2.32 zeigt dieselbe Registerstruktur wie in Bild 2.31, jetzt aber mit Steuersignalen, den Steuer-Variablen. Die Steuer-Variablen werden von einer Ablaufsteuerung erzeugt. Die Ablaufsteuerung ist Teil des Steuerwerks. Ein Beispiel für die Realisierung einer Ablaufsteuerung zur Multiplikation zweier Dualzahlen wurde im Abschnitt 2.4 behandelt.

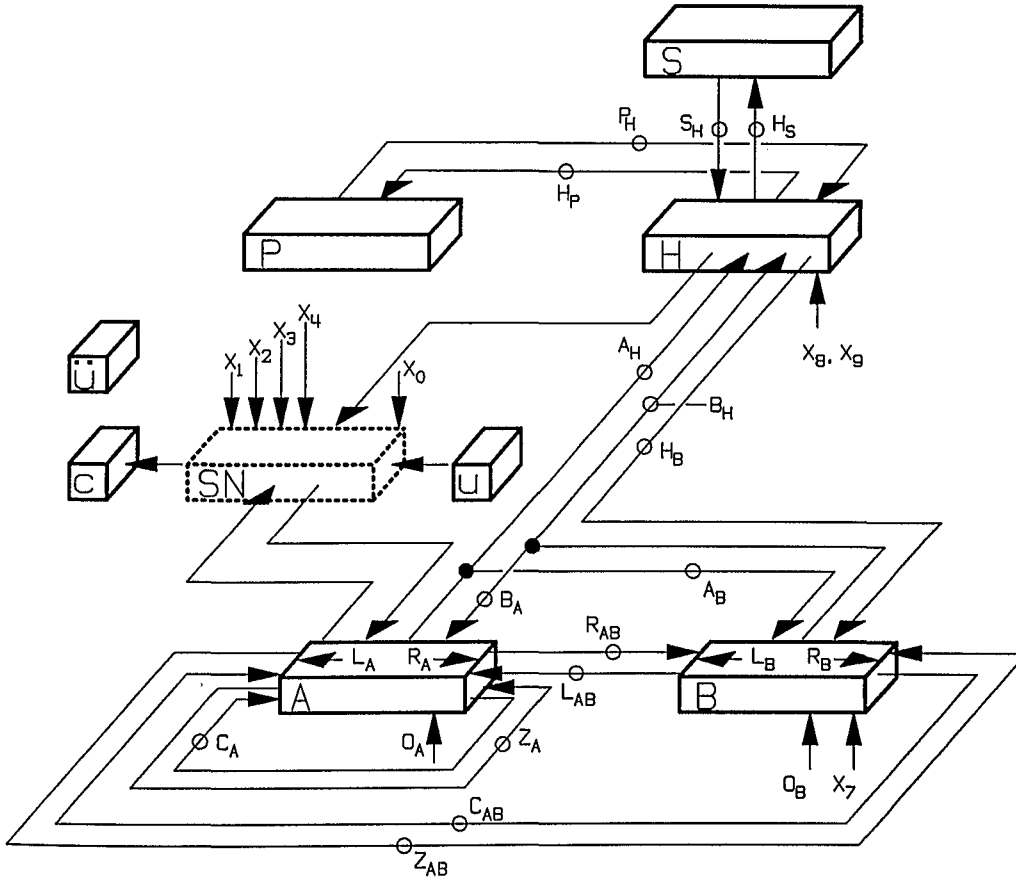


Bild 2.32: Operationswerk mit den Steuer-Variablen zur Steuerung des Datenflusses

In der nachstehenden Tabelle 2.1 sind einige wichtige Steuer-Variable für die Ausführung von Mikrobefehlen eines Operationswerkes dargestellt. Eine Befehlsanweisung läßt sich in eine Folge von derartigen Mikrobefehlen auflösen.

Tabelle 2.1: Mikrobefehle und Steuer-Variable eines Operationswerkes mit den zugeordneten Operationen

Steuer-Variable	Mikrobefehle	Operationen
$X_0 \wedge X_2 \wedge X_4$	$(A)+(H)+(u) \rightarrow A$	Addition
X_3	$(A) \wedge (H) \rightarrow A$	Konjunktion
X_2	$(A) \vee (H) \rightarrow A$	Disjunktion
$X_2 \wedge X_4$	$(A) \oplus (H) \rightarrow A$	Disvalenz
$X_1 \wedge X_2 \wedge X_3 \wedge X_4$	$\overline{(A)} \rightarrow A$	Negation
$X_3 \wedge X_4$	$0 \rightarrow A$	Löschen
$X_2 \wedge X_3$	$(H) \rightarrow A$	Transport
$L_A \wedge Z_A$	$\overleftarrow{(A)}^{a_{n-1}} \rightarrow A$	Shift. zyklisch
$L_A \wedge O_A$	$\overleftarrow{(A)}^{0} \rightarrow A$	Shift. arithmetisch
$R_A \wedge C_A$	$^{a_0}\overrightarrow{(A)} \rightarrow A$	Shift. zyklisch
R_A	$^{a_{n-1}}\overrightarrow{(A)} \rightarrow A$	Shift. arithmetisch
X_7	$0 \rightarrow B$	Löschen
A_B	$(A) \rightarrow B$	Transport
R_B	$^{b_{n-1}}\overrightarrow{(B)} \rightarrow B$	Shift. arithmetisch
$L_A \wedge L_{AB} \wedge L_B \wedge Z_{AB}$	$\overleftarrow{(AB)}^{a_{n-1}} \rightarrow AB$	Shift. zyklisch
$L_A \wedge L_{AB} \wedge L_B \wedge O_B$	$\overleftarrow{(AB)}^{0} \rightarrow AB$	Shift. arithmetisch
$R_A \wedge C_{AB} \wedge R_B \wedge R_{AB}$	$^{b_0}\overrightarrow{(AB)} \rightarrow AB$	Shift. zyklisch
$R_A \wedge R_B \wedge R_{AB}$	$^{a_{n-1}}\overrightarrow{(AB)} \rightarrow AB$	Shift. arithmetisch
X_8	$\overline{(H)} \rightarrow H$	Negation
X_9	$0 \rightarrow H$	Löschen
A_H	$(A) \rightarrow H$	Transport
H_S	$(H) \rightarrow S$	Transport
. .	. .	.
. .	. .	.

Das skizzierte Beispiel in Bild 2.33 soll verdeutlichen, wie hardwaremäßig Mikrobefehle erzeugt und damit der Datenfluß auf Registertransferebene gesteuert wird. Durch eine Folge von Mikrobefehlen kann ein bestimmter Befehl ausgeführt werden. Die Folge von Mikrobefehlen wird durch eine Ablaufsteuerung mit Steuer–Variablen realisiert. Bild 2.33 zeigt ausschnittweise die Befehlsausführung in einem Prozessor durch sog. Mikroprogramme. Die Befehlsausführung startet in einem bestimmten Anfangszustand und endet in einem definierten Zustand. Vor der Befehlsausführung müssen in entsprechenden Speicherzyklen die erforderlichen Operanden bereit gestellt werden. Ein Speicherzyklus wird wiederum von einer Ablaufsteuerung kontrolliert.

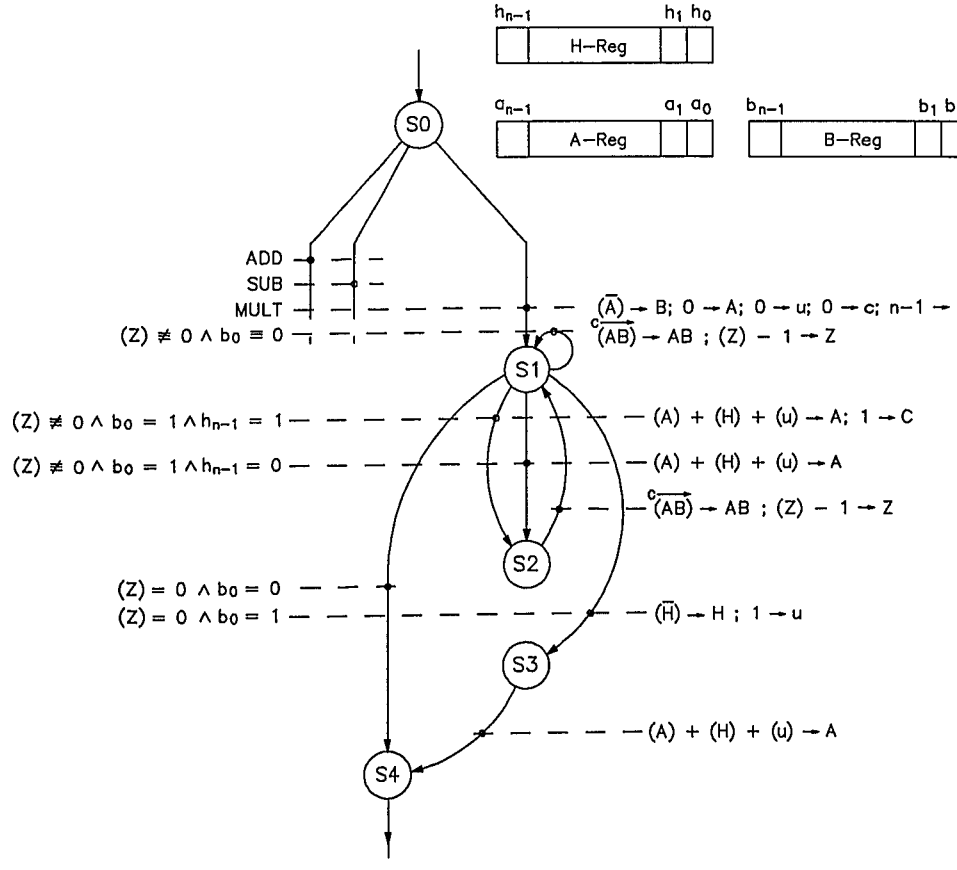
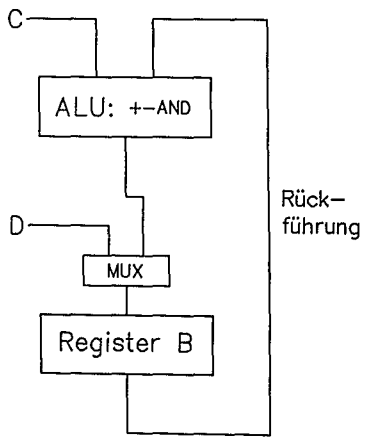


Bild 2.33: Beispiel: Ablauf eines Multiplikationsbefehls mit Mikrobefehlen

Mit einer geeigneten Hardwarebeschreibungssprache läßt sich der Datentransfer einer Blockfunktion auf Registertransferebene mit den entsprechenden Entscheidungs–Variablen und Steuer–Variablen beschreiben.

2.5.2 Zur algorithmischen Beschreibung

Im Gegensatz zur Registertransferbeschreibung ist auf der algorithmischen Beschreibungsebene eine Schaltungsstruktur noch nicht festgelegt. Den Unterschied verdeutlicht Bild 2.34. Während das algorithmische Verhalten im gewählten einfachen Beispiel in einer Zeile formuliert werden kann, nimmt die Registertransferbeschreibung Bezug auf eine gewählte Registerstruktur. Ziel der algorithmischen Synthese ist es beispielsweise eine Konfiguration zu finden, die mit einer minimalen Registerzahl auskommt.



Algorithmische Beschreibung:

B = C + D

Registertransfer–beschreibung:

B = D

B = B + C

Bild 2.34: Einfaches Beispiel einer Registertransferbeschreibung

Im Algorithmus kann auch festgelegt werden, wie die Ausgänge auf Ereignisse an den Eingängen und in Abhängigkeit von den inneren Zuständen (Entscheidungsvariable) reagieren. In einem Zeitmodell werden die Verzögerungszeiten ermittelt, mit denen die Ausgangszustände zeitlich verzögert reagieren, um auch das Zeitverhalten des Funktionsblockes richtig zu beschreiben.

Es genügt nicht, einen Funktionsblock auf algorithmischer Ebene zu beschreiben und zu verifizieren, man muß schließlich eine diese Funktion realisierende Schaltungsstruktur finden. Die Methodik der algorithmischen Synthese ist heute in vielen Bereichen ein Forschungsthema. Hierzu sei auf weiterführende Literatur verwiesen z.B. /8/.

2.5.3 Hardwarebeschreibungssprachen

Die algorithmische Beschreibung einer Logikfunktion ist nur mit einer geeigneten Hardwarebeschreibungssprache möglich. Die Formulierung des Algorithmus erfolgt in der Regel in textueller Form. Dies erfordert ein erhebliches Umdenken für den Schaltungsentwickler, da er bisher gewohnt war, die Schaltungsstruktur seiner Funktionsmodule grafisch mit Schaltungsprimitiven darzustellen. Hardwarebeschreibungssprachen haben ihren Ursprung in Programmiersprachen wie PASCAL, C oder ADA. Damit können Verhaltensmodelle für Funktionsblöcke mit unterschiedlichem Detaillierungsgrad von der Systemebene, der Registertransferebene bis zur Logikebene gebildet werden. Als IEEE-Standard wurde die Hardwarebeschreibungssprache VHDL (VHSIC-Hardware Description Language) eingeführt. In VHDL ist ein ENTITY-Konzept realisiert. Die Schnittstelle des Funktionsblocks wird in der ENTITY-Declaration beschrieben. Die eigentliche Realisierung der Architektur erfolgt in der ARCHITECTURE-Declaration. Die Formulierung ist ähnlich der von PASCAL oder ADA. Im Vergleich zu einer herkömmlichen Programmiersprache lassen sich auch nebenläufige Aktionen und das Timing-Verhalten von Grundelementen beschreiben.

Anhand eines einfachen Beispiels in Form eines Multiplexers (Bild 2.35) soll die Darstellung und die Verhaltensbeschreibung eines Funktionsblocks mit VHDL veranschaulicht werden. Herkömmlich wird ein Funktionsblock symbolisch mit Schaltungsprimitiven dargestellt (Bild 2.35). Diese symbolische Designstruktur läßt sich auch mit VHDL beschreiben (Tabelle 2.2). Das Verhalten der verwendeten Schaltungsprimitiven ist in der Verhaltens-Beschreibung (Tabelle 2.3) festgelegt. Dort wird auch das reale Timing-Verhalten berücksichtigt. Bei der Schaltungsverifikation sollen mehrere Realisierungsalternativen überprüft werden können. Die Auswahl einer Alternative zur Beschreibung des Funktionsblockes wird in der CONFIGURATION-Declaration festgelegt (Tabelle 2.4). Im konkreten Beispiel beschreibt z.B. das nand_gate-Modell gekennzeichnet durch den Namen "behavioral" die Gatter G1,G2 und G3. Diese einfache Schaltungsstruktur mit den Gattern G0,G1,G2 und G3 bildet einen Funktionsblock, der durch ein VHDL-Modell beschrieben wird.

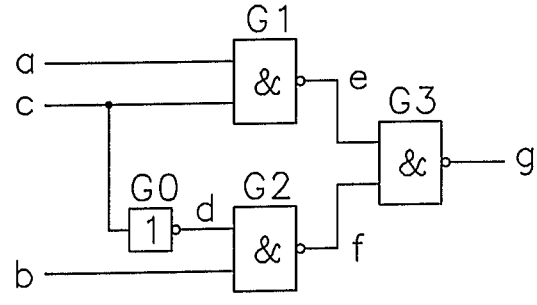


Bild 2.35: Symbolische Darstellung eines Funktionsblocks mit Schaltungsprimitiven

Tabelle 2.2: Designstruktur des Funktionsblockes in VHDL

```
ENTITY mux IS
    PORT (a,b,c: IN BIT; g: OUT BIT);
END mux;

ARCHITECTURE structural OF mux IS
    COMPONENT inv_gate PORT (a:IN BIT;y:OUT BIT);END COMPONENT;
    COMPONENT nand_gate PORT (a,b:IN BIT;y:OUT BIT);END COMPONENT;
    SIGNAL d,e,f: BIT;
BEGIN
    g0:inv_gate PORT MAP(c,d);
    g1:nand_gate PORT MAP(a,c,e);
    g2:nand_gate PORT MAP(d,b,f);
    g3:nand_gate PORT MAP(e,f,g);
END structural;
```

Die Designstruktur in Tabelle 2.2 enthält die Bauteilliste, die Netzliste und die Netz- und Verbindungsliste. Damit ist die Schaltungsstruktur festgelegt. In der PORT-Anweisung werden die nach außen gehenden Schnittstellen erklärt.

Tabelle 2.3: Verhaltensbeschreibung des NAND-Gates bzw. Inverters mit VHDL

```
ENTITY nand_gate IS
    PORT (a,b: IN BIT; y: OUT BIT);
END nand_gate;
ARCHITECTURE behavioral OF nand_gate
IS
BEGIN
    y <= a NAND b AFTER  1 ns;
END behavioral;
```

```
ENTITY inv_gate IS
    PORT (a: IN BIT; y: OUT BIT);
END inv_gate;
ARCHITECTURE behavioral OF inv_gate IS
BEGIN
    y <= NOT a AFTER 1 ns;
END behavioral;
```

Durch die Verhaltensbeschreibung der Komponenten wird deren Logikverhalten und Timing-Verhalten festgelegt. Tabelle 2.3 zeigt dies für die in Bild 2.35 verwendeten Komponenten.

Tabelle 2.4: Designkonfiguration; Auswahl einer Alternative für die Simulation

```
CONFIGURATION parts OF simple IS
FOR structural
    FOR g0:inv_gate USE ENTITY work.inv_gate(behavioral);
    END FOR;
    FOR g1,g2,g3:nand_gate USE ENTITY work.nand_gate(behavioral);
    END FOR;
END FOR;
END parts;
```

Mit der in Tabelle 2.4 dargestellten Prozedur wird eine bestimmte Schaltungsrealisierung konfiguriert. Das heißt, einzelne Schaltungsmodule sind austauschbar. Dies erhöht ganz erheblich die Flexibilität bei der Verifikation bestimmter Realisierungsalternativen.

Ein weiteres einfaches Beispiel ist in Bild 2.36 dargestellt. Hier wird mit Gatterprimitiven ein RS-Flipflop beschrieben.

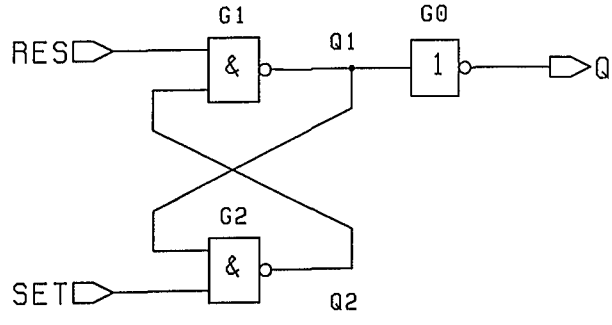


Bild 2.36: Symbolische Beschreibung der Schaltung (z.B. RS-Flipflop)

Bild 2.36 zeigt die herkömmliche symbolische Schaltungsbeschreibung. Nachstehend ist dargelegt, wie ein derartiger Funktionsblock in VHDL beschrieben wird, so daß der Funktionsblock auch simulierbar ist.

Tabelle 2.5 enthält die VHDL-Modelle der Schaltungsprimitive. Dabei ist auch das Timing-Verhalten enthalten.

Tabelle 2.5: Modelle von Schaltungsprimitiven in VHDL-Beschreibung

```
-- ***** nand_gate model *****
-- external ports
ENTITY nand_gate IS
    PORT (a,b: IN BIT; y: OUT BIT);
END nand_gate;

-- internal Behavior
ARCHITECTURE behavioral OF nand_gate IS
BEGIN
    y <= a NAND b AFTER 1 ns;
END behavioral;

-- ***** inv_gate model *****
-- external ports
ENTITY inv_gate IS
    PORT (a: IN BIT; y: OUT BIT);
END inv_gate;

-- internal Behavior
ARCHITECTURE behavioral OF inv_gate IS
BEGIN
    y <= NOT a AFTER 1 ns;
END behavioral;
```

In Tabelle 2.6 ist die Designstruktur des einfachen Beispiels dargestellt. Darin wird gezeigt wie die Schaltungsstruktur, symbolisch festgelegt in Bild 2.36, mit VHDL beschrieben wird.

Tabelle 2.6: Designstruktur eines RS-Flipflops in VHDL-Beschreibung

```
-- ***** rs flip-flop model *****
-- external ports
ENTITY rsff IS
    PORT (res,set: IN BIT; Q: OUT BIT);
END rsff;


-- internal structure
ARCHITECTURE structural OF rsff IS
    -- component types to use
    COMPONENT nand_gate
        PORT (a,b: IN BIT; y: OUT BIT); END COMPONENT;
    COMPONENT inv_gate
        PORT (a: IN BIT; y: OUT BIT); END COMPONENT;

    -- internal signals
    SIGNAL q1,q2; BIT;
BEGIN
    g0:inv_gate PORT MAP (q1,q);
    g1:nand_gate PORT MAP (res,q2,q1);
    g2:nand_gate PORT MAP (q1,set,q2);
END structural;
```

In der ENTITY-Declaration werden die Schnittstellen des Funktionsblocks nach außen festgelegt; in der ARCHITECTURE-Declaration erfolgt die Festlegung der inneren Schaltungsstruktur. Verwendet werden dabei die Modelle, definiert in Tabelle 2.5.

Nun soll diese Darstellung eines RS-Flipflops für die Schaltungssimulation verwendet werden. Dazu benötigt man u.a. die Festlegung der Testsignale. In Tabelle 2.7 ist die Festlegung von Testsignalen für das RS-Flipflop dargestellt. Die ENTITY-Declaration "rsff_stim" definiert einen Stimulus-Vektor für das RS-Flipflop.

Tabelle 2.7: Teststimuli für das RS-Flipflop

```
ENTITY rsff_stim IS
    PORT  (res,set:  OUT  BIT;  q:  IN  BIT);
END rsff_stim;


ARCHITECTURE behavioral OF rsff_stim IS
BEGIN
    set <= '1' AFTER 0 ns,
           '1' AFTER 10 ns,
           '0' AFTER 20 ns,
           '1' AFTER 30 ns;

    res <= '0' AFTER 0 ns,
           '1' AFTER 10 ns,
           '1' AFTER 20 ns,
           '1' AFTER 30 ns;
END behavioral;
```

t/nsec	set	res
0	1	0
10	1	1
20	0	1
30	1	1

Die konkrete Auswahl der Alternative zur Darstellung des RS-Flipflops erfolgt beispielhaft in Tabelle 2.8.

Tabelle 2.8: Auswahl einer Alternative zur Darstellung eines RS-Flipflops.

```
-- design management/configuration
CONFIGURATION parts OF rsff IS
FOR structural
    -- use behavioral architecture for gates g1 and g2
    FOR g1,g2:nand_gate
        USE ENTITY work.nand_gate(behavioral); END FOR;

    -- use behavioral architecture for gate g0
    FOR g0:inv_gate
        USE ENTITY work.inv_gate(behavioral); END FOR;
END FOR;
END parts;
```

Für die Verifikation der Schaltungsfunktion muß der Stimuli–Generator (generator) und der Schaltkreis (circuit) festgelegt werden. Tabelle 2.9 zeigt dies anhand unseres kleinen Beispiels.

Tabelle 2.9: RS–Flipflop–Test–Bench für die Verifikation

```
ENTITY rsff_bench IS
END rsff_bench;

ARCHITECTURE structural OF rsff_bench IS
    COMPONENT rsff_stim PORT (res,set: OUT BIT; q: IN BIT); END COMPONENT;
    COMPONENT rsff PORT (res,set: IN BIT; q: OUT BIT); END COMPONENT;
    SIGNAL res,set,q: BIT;
BEGIN
    generator:rsff_stim PORT MAP(res,set,q);
    circuit:rsff PORT MAP(res,set,q);
END structural;

CONFIGURATION parts OF rsff_bench IS
FOR structural
    FOR generator:rsff_stim USE ENTITY work.rsff_stim(behavioral);
    END FOR;
    FOR circuit:rsff USE ENTITY work.rsff(structural);
        FOR structural
            FOR g1,g2:nand_gate USE ENTITY work.nand_gate(behavioral);END FOR;
            FOR g0:inv_gate USE ENTITY work.inv_gate(behavioral); END FOR;
        END FOR;
    END FOR;
END FOR;
END parts;
```

Das Ergebnis der Verifikation des Schaltungsverhaltens ist in Bild 2.37 dargestellt. Dazu ist ein Schaltungssimulator erforderlich, der die VHDL–Beschreibung interpretieren kann.

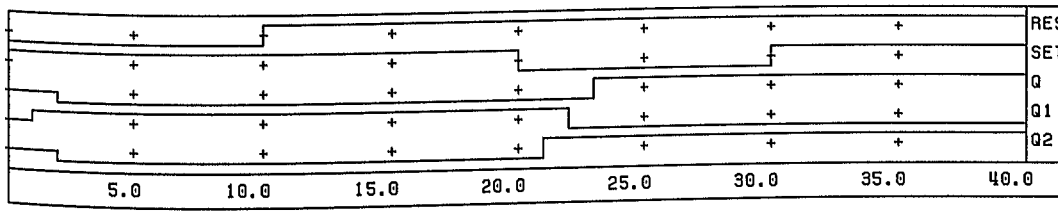


Bild 2.37: Ergebnis des Zeitverhaltens der Verifikation eines RS–Flipflops

Die beiden bisher betrachteten Beispiele (Multiplexer und RS-Flipflop) zeigen die Anwendung der VHDL-Beschreibung auf Logikebene für sehr einfache Funktionen. Der Vorteil der Hardwarebeschreibungssprache liegt aber gerade darin komplexe Schaltungsfunktionen zu beschreiben. Tabelle 2.10 zeigt die VHDL-Darstellung eines Barcode-Prozessors /16/. Der Barcode-Prozessor dient zum schnellen Auswerten des durch einen Laser-Scanner oder durch eine CCD-Kamera aufgenommenen Strichcodes. Der Sensor liefert im gewählten Beispiel das Videosignal VIDEO; das Steuersignal SCAN kennzeichnet den Scan-Mode. Realisiert werden kann der Barcode-Prozessor z.B. durch einen Gate-Array-Baustein. Eine Software-Realisierung basierend auf Standard-Mikroprozessoren wäre vergleichsweise langsam. Der Barcode-Prozessor enthält weniger als 1000 Gatteräquivalente; er hat folgende Aufgaben zu erfüllen:

* Messen und Ausgeben der Breite aller Balken,
  DATA: Breite des aktuellen Balkens,
  ADDR: Nummer des aktuellen Balkens,
  WRONG: kennzeichnet eine ungültige Ausgabe;

* Zählen der Anzahl der Balken, solange bis die vorgegebene Balkenzahl BKZN erreicht ist;

* Erkennen des Strichcode-Musters; mit READY wird die Erkennung signalisiert.

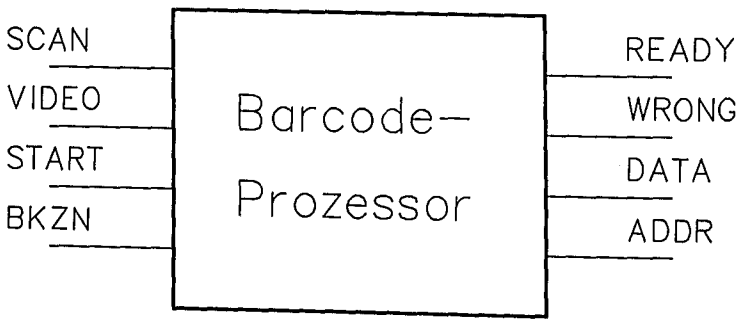


Bild 2.38: Barcode-Prozessor

Mit der in Tabelle 2.10 realisierten VHDL-Beschreibung ist der Anwender in der Lage mit einem Simulator den gewählten Algorithmus zu verifizieren und gegebenenfalls Alternativen zu erproben.

Tabelle 2.10: Funktionsmodell für den Barcode–Prozessor in VHDL–Beschreibung

```
-- Prozessor zur schnellen Mustererkennung
ENTITY barcode IS
  PORT (
    scan   :  IN  boolean;   -- Scan-Signal eines Laser-Scanners
    video  :  IN  bit;       -- Videosignal eines Laser-Scanners
    start  :  IN  boolean;   -- Start-Signal für den Prozessor
    bkzn   :  IN  integer;   -- Zahl der Balken des Strichcodes
    ready  :  OUT boolean;   -- Der Strichcode wurde erkannt
    wrong  :  OUT bit;       -- Ausgabe ist ungültig
    data   :  OUT integer;   -- Breite des aktuellen Balken
    addr   :  OUT integer;   -- Nummer des aktuellen Balken
  );
END barcode;
ARCHITECTURE sequential OF barcode IS
BEGIN
  PROCESS
    VARIABLE weiss  : integer;     -- Zählt die Breite der weißen Balken
    VARIABLE schw   : integer;     -- Zählt die Breite der schwarzen Balken
    VARIABLE bkz    : integer;     -- Zählt die Anzahl der Balken
    VARIABLE marker : bit;         -- Merker zur Erkennung eines S-W Übergangs
    CONSTANT w0     : bit := '0'; -- Weiß entspricht '0'
    CONSTANT s1     : bit := '1'; -- Schwarz entspricht '1'
  BEGIN
    ready  <= false;
    wrong <= '0';
    WAIT UNTIL start = true;
    LOOP                           -- Bis das Strichcode-Muster erkannt wurde
      WAIT UNTIL scan = true;
        marker := w0;
        bkz    := '0';
        weiss  := '0';
        schw   := '0';
        LOOP                  -- Bis eine Zählvariable (S oder W) überläuft
          IF video = w0 THEN
            weiss := weiss + 1;
            IF marker = s1 THEN
              bkz := bkz + 1;
              wrong <= '0';
            ELSE
              wrong <= '1';
            END IF;
            marker := w0;
            schw := 0;
            data <= weiss;
```

```
        ELSE
          schw = schw + 1;
          IF marker = w0 THEN
            bkz := bkz + 1;
            wrong <= '0';
          ELSE
            wrong <= '1';
          END IF;
          marker := s1;
          weiss := 0;
          data <= schw;
        END IF;
        addr <= bkz;
        EXIT WHEN (weiss = 255) OR ( schw = 255);
      END LOOP;
    EXIT WHEN (bkz = bkzn) AND ( weiss = 255);
  END LOOP;
 wrong <= '0';
 ready  <= true;                    -- Ende Meldung an die externe Steuerung
 WAIT UNTIL start = false;
 END PROCESS;
END sequential;
```

Die wenigen Beispiele dieses Abschnittes sollen die Möglichkeiten einer Hardwarebeschreibungssprache aufzeigen. Für eine ausführliche Darstellung der VHDL-Beschreibung von Logikfunktionen sei auf /14/ bis /16/ verwiesen. Das in Tabelle 2.10 behandelte Beispiel ist der algorithmischen Ebene zugeordnet. Es wird dort nicht auf eine Registerstruktur Bezug genommen. Nicht berücksichtigt werden dabei die zeitlichen Abläufe; dies müßte in einem verfeinerten Modell vorgenommen werden. Der algorithmische Ablauf läßt sich auch in einem Struktogramm darstellen. Tabelle 2.11 zeigt dies für das Beispiel Barcode-Prozessor.

Tabelle 2.11: Struktogramm für den Algorithmus des Barcode–Prozessors

ready <= FALSE wrong <= '0'	d.h. noch kein Strichcode erkannt signalisiert, daß Ausgabe auf 'data' und 'addr' ungültig ist.		
Warten, bis am Eingang 'start' TRUE anliegt, d.h. Start–Signal			
warten, bis am Eingang 'scan' TRUE anliegt			
marker := w0	d.h. es wird davon ausgegangen, daß das 'video'–Signal am Anfang weiß ist		
bkz := 0 ; weiss := 0 ; schw := 0	Variableninitialisierung		
'video'–Signal = Weiß–Pegel ?			
J	N		
weiß := weiß+1 = Breite der weißen Balken		schw := schw +1 = Breite der schwarzen Balken	
marker = s1 ?		marker = w0 ?	
J	N	J	N
d.h. beim letzten Durchlauf war 'video'–Signal schwarz bkz := bkz +1 Balkenanzahl erhöhen wrong <= '0'	d.h. beim letzten Durchlauf war 'video'–Signal weiß wrong <= '1'	d.h. beim letzten Durchlauf war 'video'–Signal weiß bkz := bkz +1 Balkenanzahl erhöhen wrong <= '0'	d.h. beim letzten Durchlauf war 'video'–Signal schwarz wrong <= '1'
marker = w0 d.h. 'video'–Signal speichern		marker = s1 d.h. 'video'–Signal –Pegel speichern	
schw := 0 Breite des letzten schwarzen Balkens löschen		weiß := 0 Breite des letzten weißen Balkens löschen	
data <= weiß d.h. am 'data' –Ausgang wird die Breite des weißen Balkens angezeigt (hochgezählt)		data <= schwarz d.h. am 'data' –Ausgang wird die Breite des schwarzen Balkens angezeigt (hochgezählt)	
addr <= bkz	Ausgabe der bislang gezählten Balken (Summe aus schwarzen + weißen Balken)		
Schleife sooft durchlaufen, bis Balkenbreite weiß oder schwarz 255 erreicht hat			
Schleife sooft durchlaufen, bis Zahl der Balken erreicht und Balkenbreite weiß 255 ist			
wrong <= '0' ready <= TRUE	Ausgabe auf 'data' und 'addr' wieder ungültig d.h. Strichcode wurde erkannt		
warten, bis am Eingang 'start' FALSE anliegt			
weiter am Struktogramm–Anfang!			

2.5.4 Zum Schaltungsentwurf mit CAE-Workstations

In zunehmendem Maß werden höhere Abstraktionsebenen bei der Entwicklung durch Entwurfswerkzeuge unterstützt. Bild 2.39 zeigt die bisherige Vorgehensweise bei der Entwurfsdarstellung und Entwurfsverifikation auf einer CAE-Workstation. Zunächst wird die symbolische Schaltungsstruktur bis auf Gatterebene aufgebaut, anschließend verifiziert. Nach erfolgreicher Verifikation der Logikfunktion wird die geometrische Schaltungsstruktur, das Layout, erstellt. Zur Vertiefung dieses Abschnittes sei auf /17/, /18/ verwiesen.

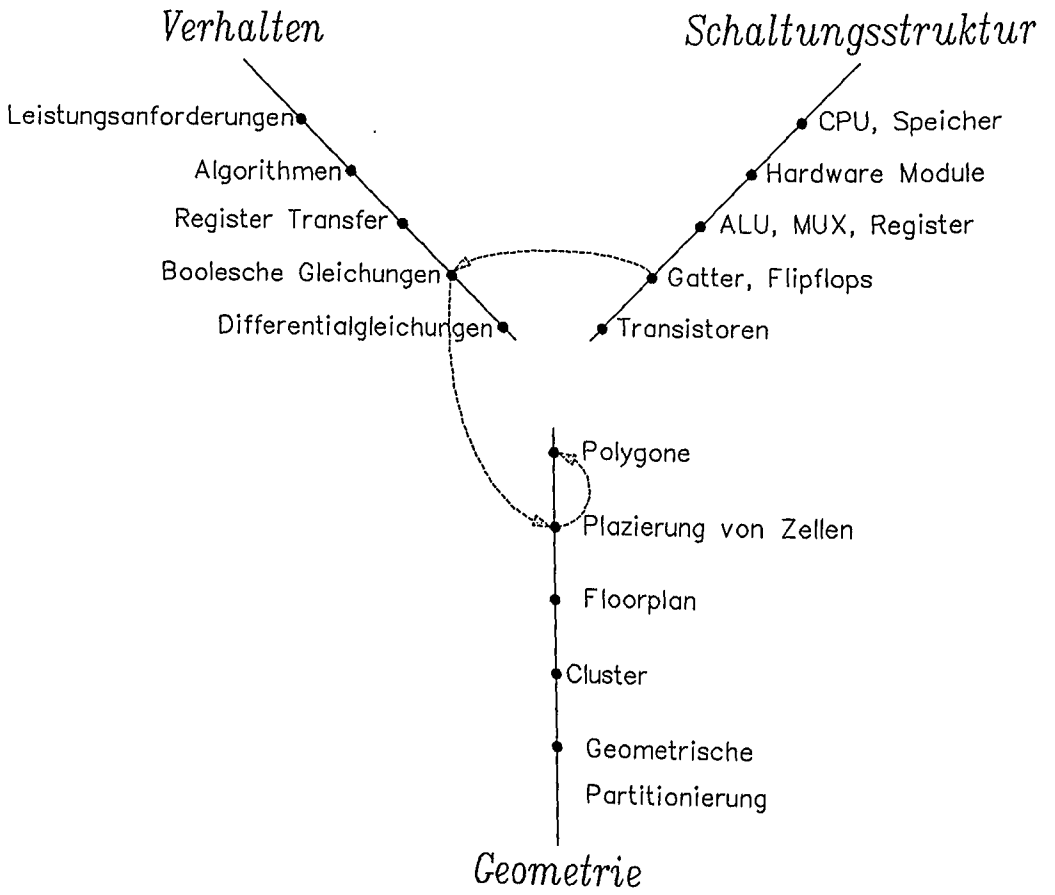


Bild 2.39: Erste Generation eines CAE-Systems

In einer Weiterentwicklung der Vorgehensweise nach Bild 2.39 wird auf Registertransferebene zunächst die Registerstruktur und deren Verknüpfung symbolisch beschrieben (Bild 2.40). Nach erfolgreicher Verifikation des Logikverhaltens mit einer Registertransferbeschreibung wird eine Vorplazierung der Blockfunktionen auf geometrischer Ebene (Floorplanning) vorgenommen. Die Blockfunktionen werden dann aufgelöst bis auf Gatterebene und herkömmlich mit einem Logiksimulator verifiziert. Nach erfolgreicher Verifikation wird das komplette Layout erstellt.

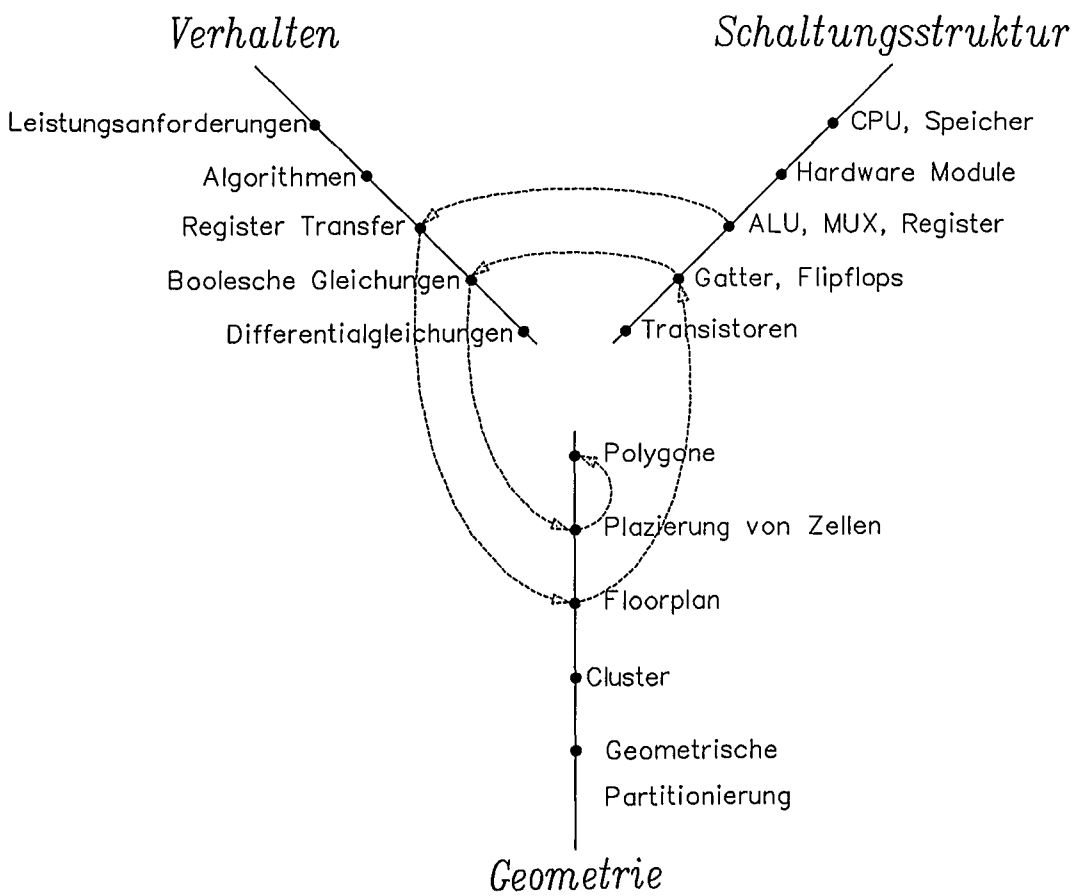


Bild 2.40: Zweite Generation eines CAE-Systems

Künftig wird bei CAE-Systemen auch eine algorithmische Beschreibung und algorithmische Synthese von Logikfunktionen mit einer geeigneten Hardware-beschreibungssprache verstärkt möglich sein (Bild 2.41).

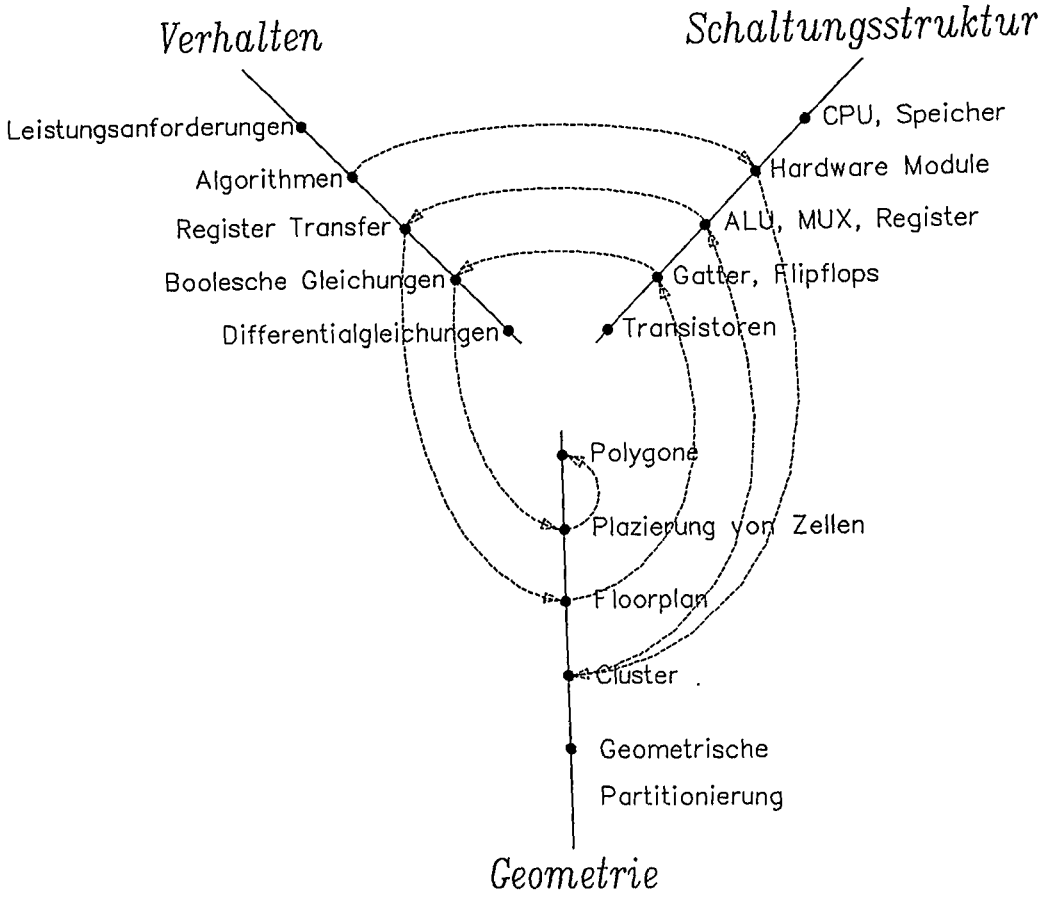


Bild 2.41: Dritte Generation eines CAE-Systems

Um aus einer Architektur-Beschreibung eine Logiksynthese vornehmen zu können, muß sichergestellt sein, daß diese Beschreibung auch in Hardware umsetzbar ist. Allgemeine Hardwarebeschreibungssprachen erlauben auf algorithmischer Ebene oder auf Architektur-Ebene Beschreibungen, die formal zwar richtig sind, aber nicht automatisch bzw. nicht optimal in Hardware umsetzbar sind.

Bei der Vorgehensweise zur Verifikation einer Logikfunktion nach Bild 2.39 muß die Schaltungsstruktur vor der Logiksimulation bis auf Gatterebene festgelegt werden. Zeigt die Logiksimulation ein Fehlverhalten der Schaltung, so muß der Entwurf verifiziert und wiederum bis auf Gatterebene aufgelöst werden, um eine erneute Logiksimulation vornehmen zu können. Dies entspricht der Vorgehensweise einer Bottom-Up-Entwurfsmethode (Bild 2.42). Bei einer Schaltungsverifikation schon auf algorithmischer Ebene bzw. Registertransferebene werden frühzeitig Fehler erkannt. Es können zudem leicht Realisierungsalternativen erprobt werden. Bild 2.43 zeigt die Vorgehensweise nach der Top-Down-Entwurfsmethode mit Verifikationsmöglichkeiten in einem sehr frühen Entwurfsstadium. Je früher ein Fehler erkannt wird, um so kostengünstiger ist die Fehlerbehebung.

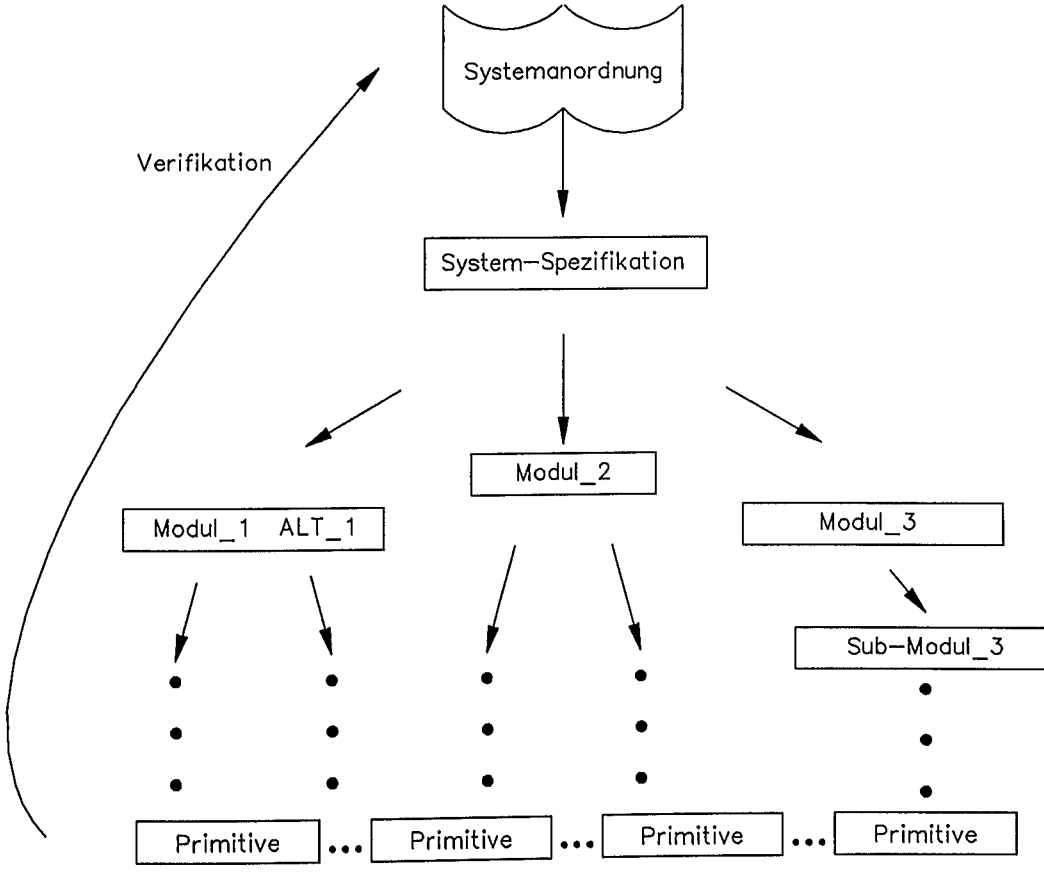


Bild 2.42: Entwurfsverifikation nach der Bottom-Up-Methode

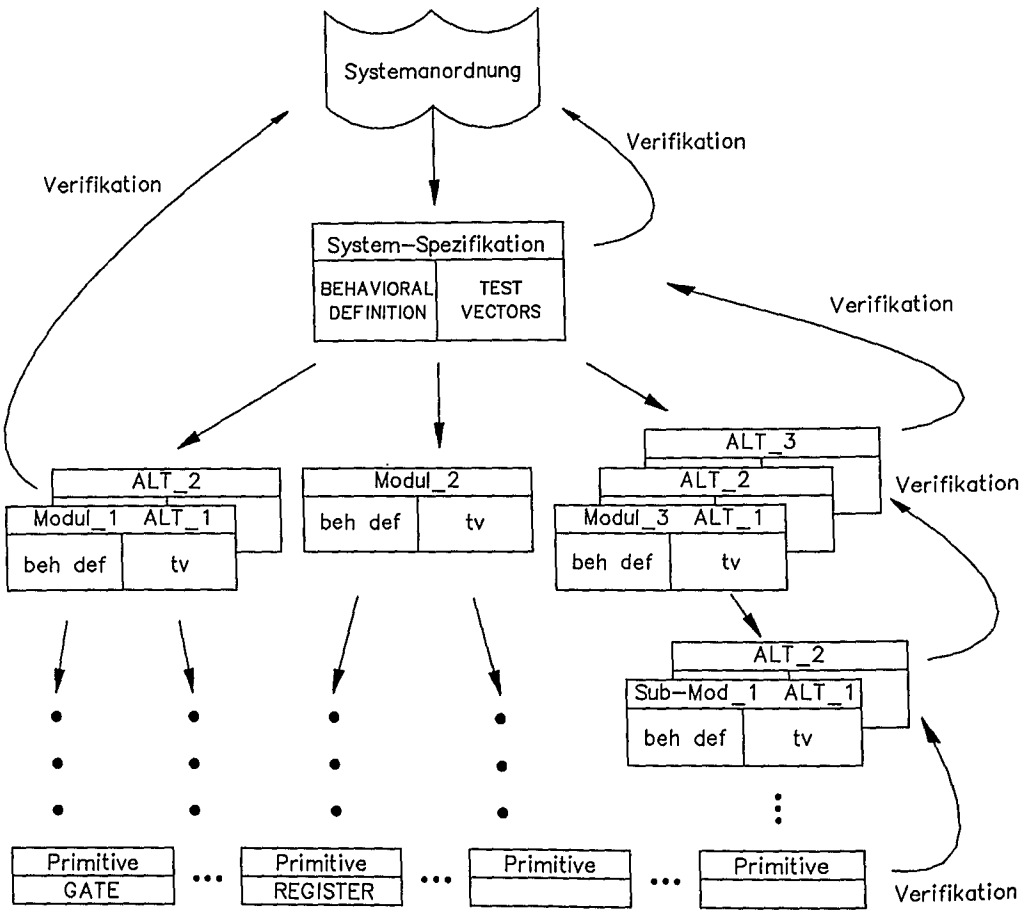


Bild 2.43: Entwurfsverifikation nach der Top–Down–Methode

Wie bereits dargelegt, kann mit der CONFIGURATION–Declaration eine Design–Alternative für einen Funktionsblock mit der Hardwarebeschreibungssprache VHDL festgelegt werden. Dies ermöglicht die Auswahl mehrerer Entwurfsalternativen in einer sehr frühen Entwurfsphase.

Bild 2.44 zeigt die Entwurfsdarstellung und Entwurfsverifikation auf CAE-Workstations. Der hierarchische Aufbau einer Schaltungsstruktur erfolgt im allgemeinen durch die *grafische Schaltplaneingabe*. Die Schaltungselemente können aus einer Bibliothek von Funktionsmodulen entnommen werden; neue Funktionsmodule sind u.a. über Synthesewerkzeuge zu erstellen. Mit der *DO_SIM*-Funktion wird das Schaltungsverhalten für vorgegebene Testvektoren ermittelt und verifiziert. Dazu benötigt man Simulationsmodelle, die aus der Bibliothek B entnommen werden. Erfüllt das elektrische/logische Verhalten die gestellten Anforderungen, so wird mit der *DO_LAYOUT*-Funktion das geometrische Layout erstellt. Die dafür notwendigen geometrischen Bauformen sind in der Bibliothek C enthalten.

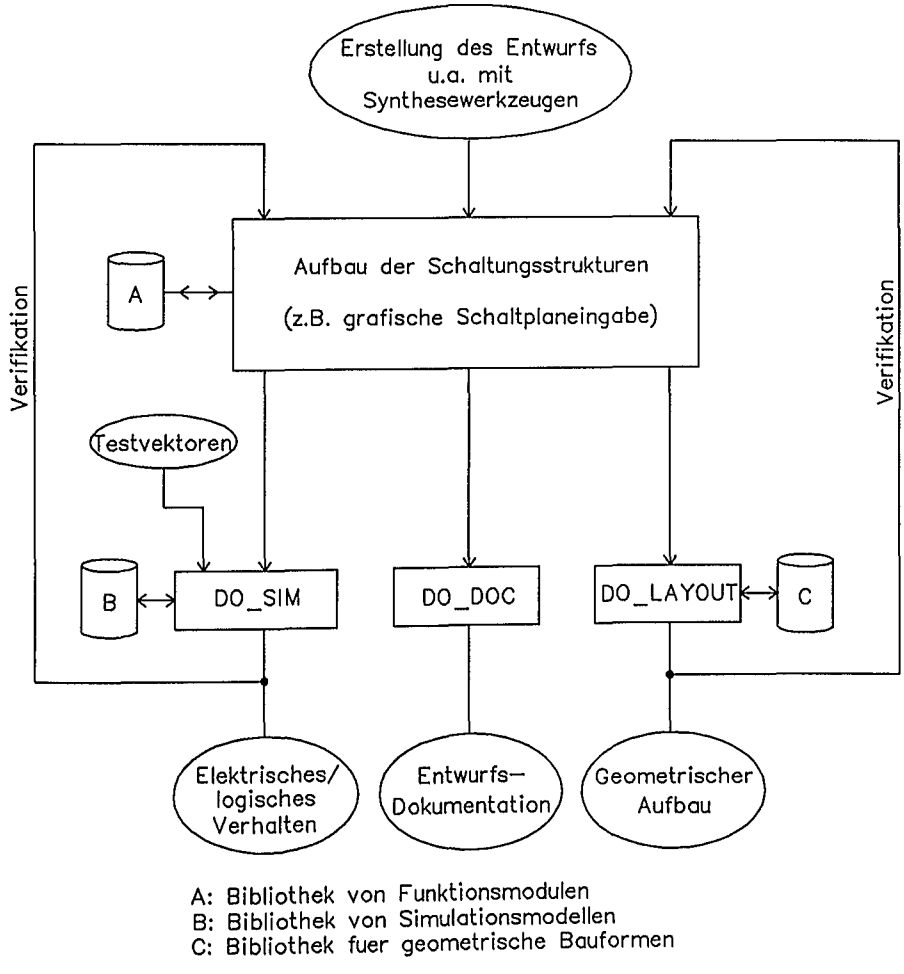


Bild 2.44: Zur Entwurfsdarstellung und Entwurfsverifikation mit CAE-Workstations

Literatur zu Kapitel 2

/1/ W. K. Giloi: Rechnerarchitektur. Springer, 1981

/2/ M. A. Breuer (Hrsg.): Digital System Design Automation: Languages, Simulation & Data Base. Pitman, 1975

/3/ F. J. Rammig: Systematischer Entwurf digitaler Systeme. Teubner, Stuttgart, 1989

/4/ F. J. Hill, G. R. Petterson: Digital Systems: Hardware Organization and Design. John Wiley & Sons, 1978

/5/ R. M. Kline: Structured Digital Design. Prentice-Hall, 1983

/6/ C. Mead; L. Conway: Introduction to VLSI-Systems. Addison Wesley, 1980

/7/ B. Preas, M. Lorezetti (Hrsg.): Physical Design Automation of VLSI Systems. Benjamin Cummings Publishing Company, 1988

/8/ R. Kolla, P. Molitor, H.-G. Osthof: Einführung in den VLSI-Entwurf. Teubner, Stuttgart, 1989

/9/ G. De. Micheli, A. Sangiovanni-Vincentelli, P. Antognetti (Hrsg.): Digital System for VLSI Circuits-Logic Synthesis and Silicon Compilation. Martinus Nijhoff Publisher, 1987

/10/ D. Gajski (Hrsg.): Silicon Compilation. Addison Wesley, 1988

/11/ J. Bortolazzi, K.D. Müller-Glaser: An Approach to Computer Aided Specification. Lehrstuhl Rechnergestützter Schaltungsentwurf, Universität Erlangen, 1989

/12/ D. Gajski, R. Kuhn: Guest Editors' Introduction: New VLSI Tools. Computer, December 1983

/13/ J. Duley, D. Dietmeyer: A Digital Design Language (DDL). IEEE Transactions on Computers, Vol. C-24, No. 2, 1975

/14/ IEEE-Std 1076-1987: IEEE Standard VHDL Language Reference Manual, March 1988

/15/ D.R. Coelho: The VHDL Handbook. Kluwer Academic Publishers; Boston, Dordrecht, London, 1989

/16/ E. Hörbst, u.a.: Synthese – Entwurfsmethode der Zukunft. Elektronik, Heft 23, 1989

/17/ T. Ohtsuki (Hrsg.): Advances in CAD for VLSI, Band 1–7. North Holland 1986/87

/18/ S. M. Trimberger: An Introduction to CAD for VLSI. Kluwer, 1987

Fragen zu Kapitel 2

F2.1 Skizzieren Sie den grundsätzlichen Aufbau eines Elektroniksystems!

F2.2 Welche Punkte sollte eine Aufgabenbeschreibung in Form eines Pflichtenheftes enthalten?

F2.3 Nennen Sie wichtige Punkte zur Festlegung einer Systemspezifikation.

F2.4 Was sind die Aufgaben und Ziele der Entwurfsautomatisierung?

F2.5 Skizzieren Sie das Grundschema zur Schaltungssynthese!

F2.6 Wodurch unterscheidet sich die Vorgehensweise zur Realisierung eines Softwaremoduls von der zur Realisierung eines Hardwaremoduls?

F2.7 Was kann ein Silicon–Compiler?

F2.8 Was versteht man unter einem hierarchischen Entwurfskonzept; wie sollte der Entwurf gegliedert sein?

F2.9 Nennen Sie wichtige Grundsätze zum Systementwurf!

F2.10 Skizzieren Sie die Entwurfsdarstellungsarten im sog. "Y–Diagramm"!

F2.11 Was versteht man unter prozeduraler Verhaltensbeschreibung eines Funktionsblocks?

F2.12 Welche Beschreibungsebenen enthält der Schaltungsstrukturbereich?

F2.13 Was versteht man unter "Floorplanning" im Geometriebereich?

F2.14 Beschreiben und skizzieren Sie das Grundkonzept zur Realisierung von Logikfunktionen mit endlichen Zustandsautomaten.

F2.15 Nennen Sie Beispiele zur Realisierung von Zustandsautomaten; entwickeln Sie eine Ablaufsteuerung für einen Additions-Befehl zweier Dualzahlen.

F2.16 Was versteht man unter einer Registertransferbeschreibung; welche wesentlichen Elemente enthält eine Schaltungsstruktur auf Registertransferebene?

F2.17 Es ist eine Ablaufsteuerung für die Zweiweggleichrichtung (Betragsbildung) einer Folge von 8-Bit-Datenwörtern mit einem Zustandsautomaten zu realisieren (Ladephase, Signalverarbeitungsphase, Ladephase, Abschluß sind zeitlich getrennt);

Steuer-Eingangssignale:

START signalisiert das Anliegen eines neuen Datenworts;
RESET bringt die Ablaufsteuerung von jedem Zustand aus in einen definierten Anfangszustand;

Entscheidungs-Variable:

MSB Vorzeichen des Datenworts;

Steuer-Ausgangssignale:

LA Load Eingangsregister;
LB Load Ausgangsregister;
INV Anweisung an die ALU zur Invertierung;
READY Abschluß der Signalverarbeitung.

F2.18 Skizzieren Sie für die vorherige Aufgabenstellung eine geeignete Registertransferstruktur.

F2.19 Wodurch unterscheiden sich CAE-Systeme der ersten, zweiten und dritten Generation?

F2.20 Welche Voraussetzungen müssen bei der Entwurfsverifikation nach der Top-Down-Methode gegeben sein?

F2.21 Welche grundlegenden Sprachkonstrukte liegen der Hardware-beschreibungssprache VHDL zugrunde?

F2.22 Welche wesentlichen Elemente enthält die Verhaltensbeschreibung eines Funktionsmoduls mit VHDL?

F2.23 Wodurch unterscheidet sich die algorithmische Beschreibung von der Registertranfserbeschreibung?

F2.24 Wie sieht die heute übliche Praxis der Synthese, der Darstellung und Verifikation eines Entwurfs mit CAE-Workstations aus?

3. Zur Entwurfsrealisierung

3.1 CMOS-Technologie

CMOS-Technologie ist die verbreitetste Halbleitertechnologie im VLSI- oder ASIC-Bauelementebereich. Dabei werden auf einem gemeinsamen Halbleitersubstrat n-Kanal- und p-Kanal-Anreicherungs-Feldeffekttransistoren (n-FET bzw. p-FET) hergestellt und verschaltet.

Bild 3.1 zeigt den schematischen Aufbau eines p-FET. Dort ist in ein p-dotiertes Halbleitersubstrat durch Dotierung eine n-Wanne eingebracht[1]. Diese n-Wanne bildet den Platz für einen p-FET.

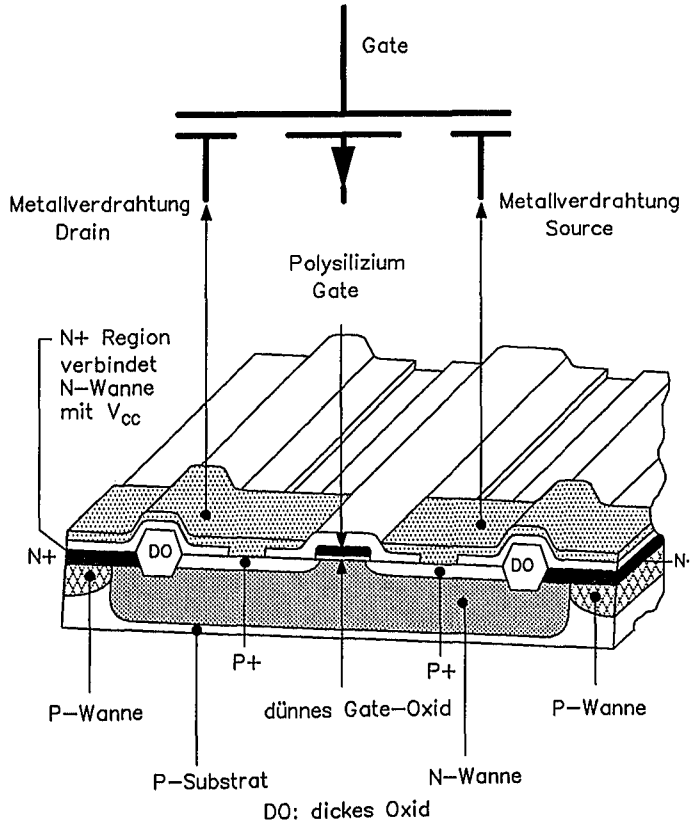


Bild 3.1: Aufbau eines p-Kanal-Anreicherungs-Feldeffekttransistors

[1] Die Beschreibung entspricht nicht der Fertigungsreihenfolge

Die Source- und Drain-Anschlüsse des p-FET werden durch Eindiffusion bzw. Ionenimplementation zweier stark dotierter p^+-Wannen hergestellt. Der kurze Rest der n-Wanne zwischen diesen Gebieten bildet den Stromkanal des p-FET. Da hier zwei pn-Übergänge gegeneinander geschaltet sind, ist dieser Kanal normalerweise nicht leitend. Der Kanal ist von einer sehr dünnen Siliziumdioxid-Schicht bedeckt, auf der leitfähiges polykristallines Silizium als Gate aufgebracht ist.

Die n-Wanne wird über stark n-dotierte Halbleitergebiete mit dem positiven Pol der Betriebsspannungsquelle verbunden. Das p-Substrat ist mit dem negativen Pol der Betriebsspannungsquelle verbunden. Der pn-Übergang zwischen Substrat und der n-Wanne ist deshalb gesperrt. Source, Drain und Gate sind mit Siliziumdioxid abgedeckt, auf dem Aluminium-Leiterbahnen zur Verdrahtung der Transistoren geführt werden. Die Kontaktierung von Source und Drain erfolgt über Löcher im überdeckenden Oxid. Die Metall-Leiterbahnen können wiederum mit Siliziumdioxid abgedeckt sein, auf das eine zweite Metallisierungsebene zu Verdrahtungszwecken aufgebracht werden kann.

Wenn das Gate mit einer negativen Spannung bezüglich der Source-Elektrode beaufschlagt wird, sammeln sich unter dem dünnen Gate-Oxid Löcher an und der Kanal wird leitend. Dies ist in den Bildern 3.2 und 3.3 anschaulich dargestellt.

Ist die Gate-Source-Spannung negativer als die Schwellenspannung $U_{th,p}$, wird die Zone zwischen den p-Gebieten (Source bzw. Drain) unterhalb des dünnen Gate-Oxids löcherleitend und ein Strom fließt zwischen Source und Drain. Typische Werte für den Betrag der Schwellenspannung liegen bei etwa 0.8 Volt.

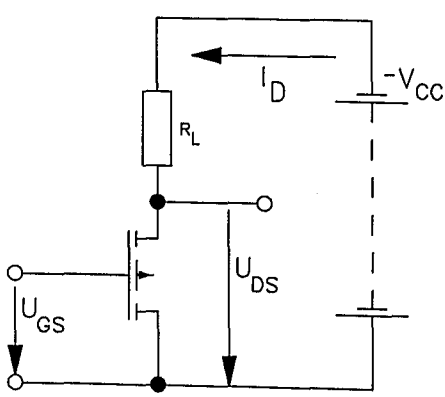


Bild 3.2: Schaltbild zur Übertragungskennlinie eines p-FET

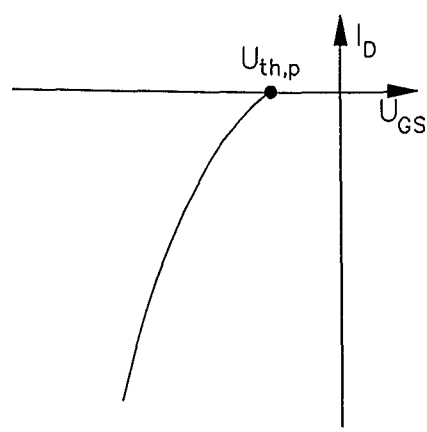


Bild 3.3: Übertragungskennlinie eines p-FET (schematisch)

Bei kleinen negativen Spannungen zwischen Drain und Source wächst der Betrag des Drainstroms mit wachsendem Betrag der Drain–Source–Spannung. (Bild 3.4) Bei Überschreiten der Sättigungsspannung $|U_{DS,sat}| = |U_{GS} - U_{th,p}|$ kommt es zur Abschnürung des Kanals, d.h. die Anreicherung mit Löchern unter dem dünnen Gate–Oxid wird in der Nähe des Drain–Anschlusses wieder rückgängig gemacht. Deshalb wächst der Betrag des Drainstroms nicht weiter an. Der Drainstrom kann mit Hilfe der Gate–Source–Spannung auch im Sättigungsgebiet gesteuert werden. Ist die Gate–Source–Spannung 0 Volt, so fließt auch praktisch kein Drainstrom. Der Widerstand zwischen Source und Drain ist sehr groß. Durch Anlegen einer Gate–Source–Spannung kann der Widerstand zwischen Source und Drain stark verkleinert werden. Im Sättigungsgebiet wirkt die Source–Drain–Strecke wie eine durch die Gate–Source–Spannung gesteuerte Stromquelle, außerhalb des Sättigungsgebietes wie ein durch die Gate–Source–Spannung steuerbarer Widerstand.

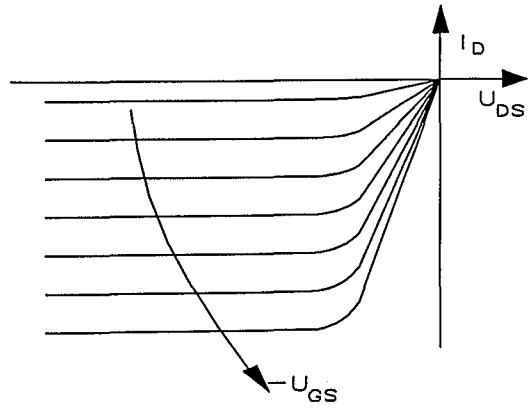


Bild 3.4: Ausgangskennlinienfeld eines p–FET (schematisch)

Die Bilder 3.5 bis 3.8 zeigen schematisch den Aufbau des n–Kanal–FET und dessen Kennlinien. Source und Drain sind hier als n^+–Wanne in einer p–Wanne ausgeführt. Positive Spannungen zwischen Gate und Source reichern Elektronen unterhalb des dünnen Gate–Oxids an und der Kanal zwischen Source und Drain wird elektronenleitend. Die Schwellenspannung $U_{th,n}$ liegt bei etwa +0,8 Volt.

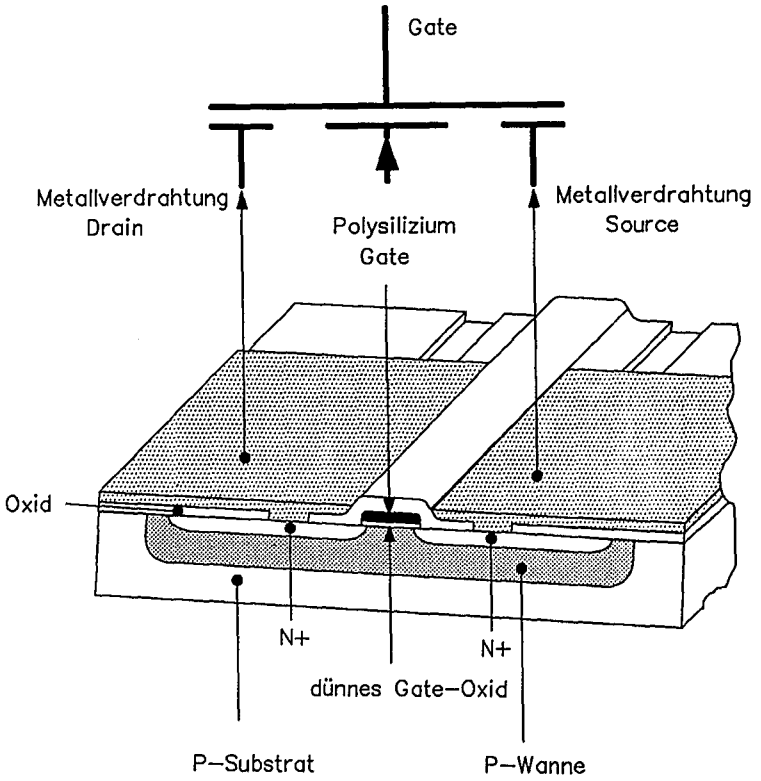


Bild 3.5: Aufbau eines n-Kanal-Anreicherungs-Feldeffekttransistors

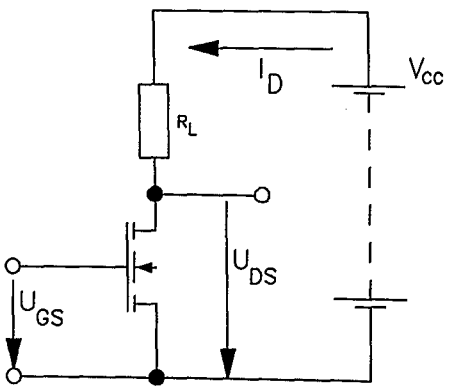


Bild 3.6: Schaltbild zur Übertragungskennlinie eines n-FET

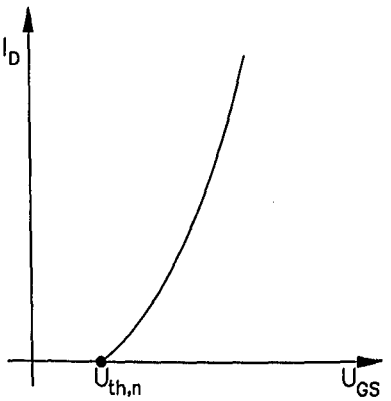


Bild 3.7: Übertragungskennlinie eines n-FET (schematisch)

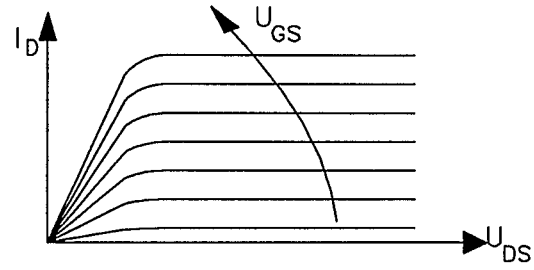


Bild 3.8: Ausgangskennlinienfeld eines n–FET (schematisch)

Wird der Arbeitswiderstand des n–FET in Bild 3.6 durch einen p–FET ersetzt, erhält man die CMOS–Grundstruktur. Der p–FET ist der aktive Arbeitswiderstand des n–FET, der n–FET ist der aktive Arbeitswiderstand des p–FET. Bild 3.9 zeigt diese Grundstruktur im Schnitt b) und einer Draufsicht a).

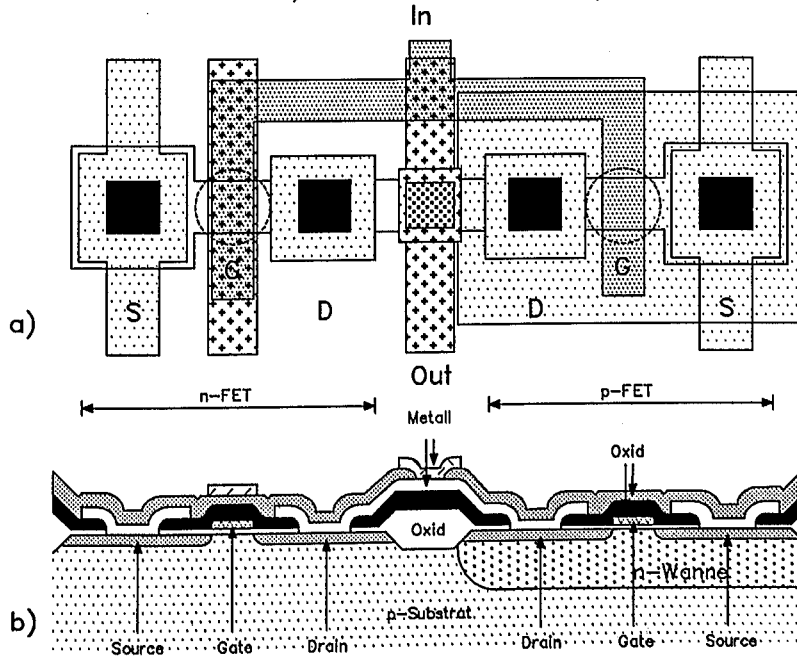


Bild 3.9: Aufbau einer CMOS–Grundstruktur, a) Layout, b) Querschnitt

In diesem Bild sind die Drain-Anschlüsse bzw. die Gates beider Transistoren bereits verbunden. Wird der Source-Anschluß des n-FET mit dem Minuspol der Spannungsquelle (Masse, V_{SS}) und der Source-Anschluß des p-FET mit dem Pluspol der Betriebsspannungsquelle (V_{CC}) verbunden, so wirkt diese Struktur wie ein Inverter (siehe Bild 3.10).

Bild 3.11 und Bild 3.12 zeigen schematisch die Übertragungskennlinie des CMOS-Inverters und die Betriebsbedingungen der beiden Transistoren in Abhängigkeit von der Eingangsspannung.

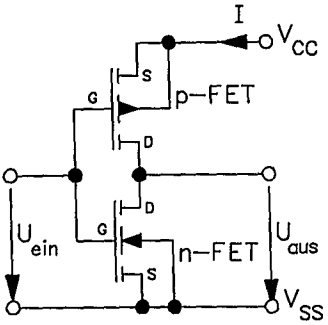


Bild 3.10: CMOS-Inverter

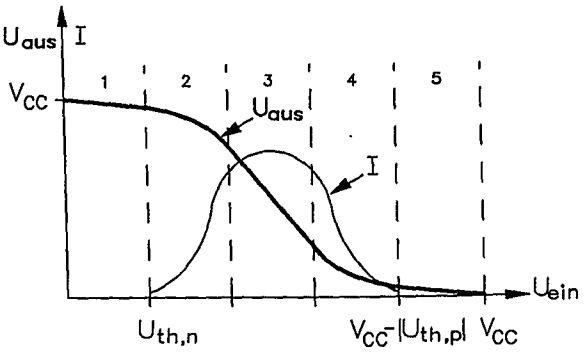


Bild 3.11: Übertragungskennlinie und Stromaufnahme eines CMOS-Inverters (schematisch, Bereiche siehe Bild 3.12)

Bereich	Bedingung	p-FET	n-FET
1	$0 \leq U_{Ein} \leq U_{th,n}$	ungesättigt	gesperrt
2	$U_{Aus} - \lvert U_{th,p} \rvert \geq U_{Ein} \geq U_{th,n}$	ungesättigt	gesättigt
3	$U_{Aus} - \lvert U_{th,p} \rvert \leq U_{Ein} \leq U_{Aus} + U_{th,n}$	gesättigt	gesättigt
4	$U_{Aus} + U_{th,n} \leq U_{Ein} \leq V_{CC} - \lvert U_{th,p} \rvert$	gesättigt	ungesättigt
5	$V_{CC} - \lvert U_{th,p} \rvert \leq U_{Ein} \leq V_{CC}$	gesperrt	ungesättigt

Bild 3.12: Betriebsbedingungen der Transistoren in Abhägigkeit von der Eingangsspannung

3.2 CMOS–Gate–Arrays

CMOS–Gate–Arrays sind Halbfabrikate, die in großen Stückzahlen vorgefertigt werden. Die kundenspezifische Funktion wird in wenigen Prozeßschritten durch Aufbringen von typischerweise zwei metallischen Verdrahtungsebenen eingebracht. Mit der Metallverdrahtung werden die Transistoren des Halbfabrikats verbunden. Bild 3.13 zeigt den Grundaufbau eines channeled Gate–Arrays. An den Rändern

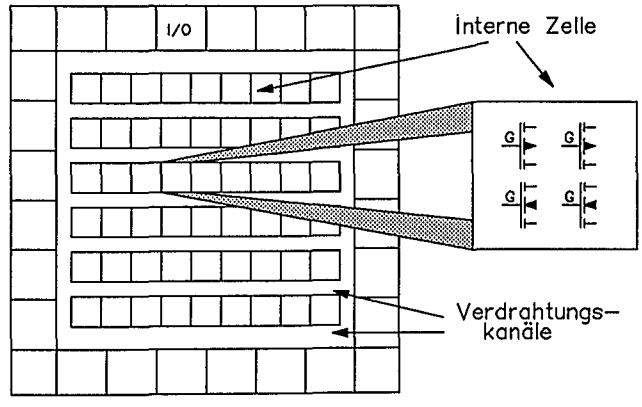


Bild 3.13: Aufbau eines "channeled" Gate–Arrays (schematisch)

des Chips sind Ein–/Ausgangszellen (I/0, äußere Zellen) mit Kontaktierungsflächen sowie Stromzuführungen angebracht. Der innere Bereich des Chips ist im Wechsel mit vorgefertigten Transistorzellen (interne Zellen) und freiem Verdrahtungsplatz (Verdrahtungskanal) regelmäßig bedeckt. Eine innere Zelle enthält üblicherweise zwei n–FET und zwei p–FET. Die Zahl der vorgefertigen inneren Zellen liegt im Bereich von einigen Hundert bis einige Zehntausend.

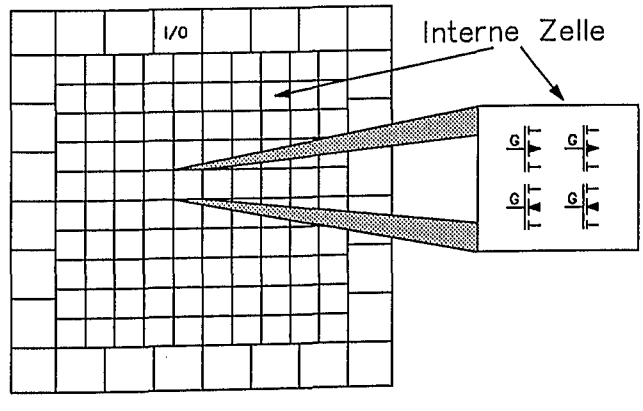


Bild 3.14: Aufbau eines "free channel" Gate–Arrays (Sea of Gates) (schematisch)

Bei den Free Channel Gate-Arrays, die auch mit Sea of Gates bezeichnet werden, ist die ganze verfügbare innere Siliziumfläche mit inneren Zellen überdeckt (siehe Bild 3.14). Zur Führung der Metallverdrahtung werden einfach innere Zellen überdeckt. Deren Transistoren sind dann nicht nutzbar. Die Zahl der verfügbaren inneren Zellen liegt zwischen einigen Tausend bis über Hunderttausend. Die Überdeckung von inneren Zellen durch die Verdrahtung begrenzt die Nutzung der verfügbaren internen Zellen auf etwa 50%, während der Nutzbarkeitsgrad bei channeled Gate-Arrays bei etwa 90% liegt. Die Begrenzung der Nutzbarkeit begünstigt die erfolgreiche automatische Verdrahtung. Bei Sea of Gates mit regulären Strukturen wie z.B. RAM, ROM oder PLA-Strukturen kann der Zell-Nutzungsgrad auch höher sein (bis etwa 75%).

Der effektive Nutzungsgrad, das ist der Anteil der internen Zellen, der für die eigentliche anwendungsspezifische Funktion verfügbar ist, kann durch erforderliche Zusatzlogik weiter reduziert werden. Solche Zusatzlogik ist häufig notwendig, um den fertigen Chip überhaupt testen zu können. Zur Sollfunktion trägt diese Zusatzlogik in der Regel nichts bei.

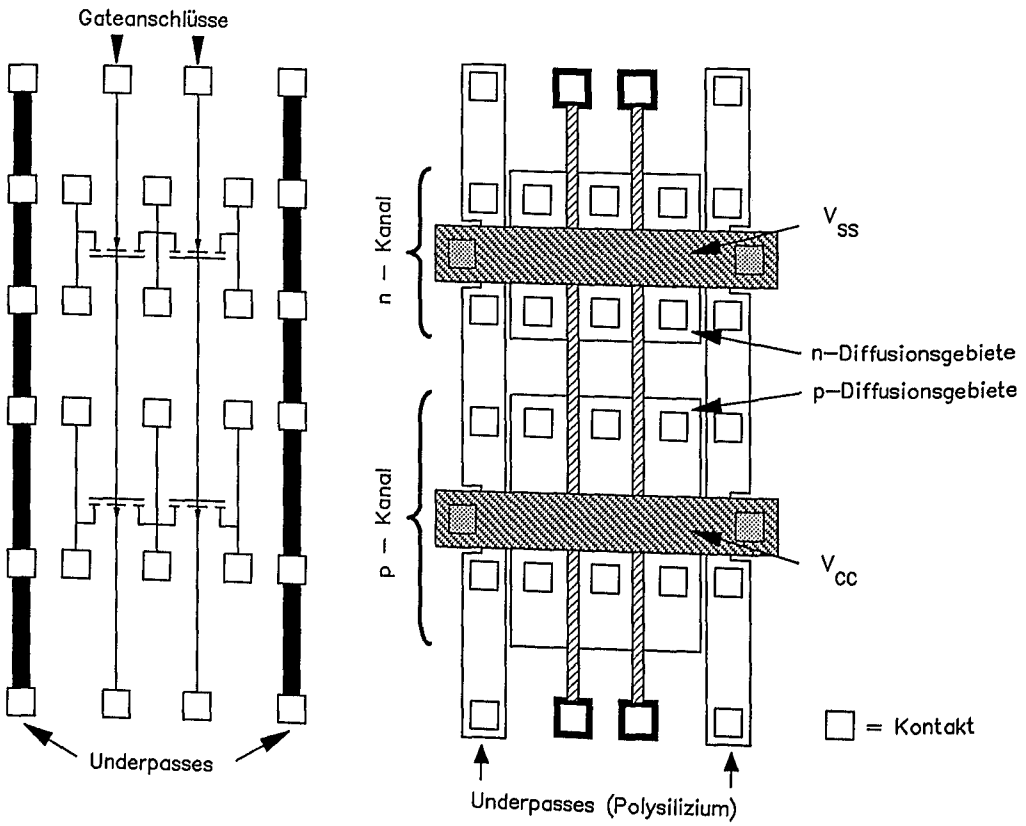


Bild 3.15: Beispiel des Aufbaues einer internen Zelle

In Bild 3.15 ist der Grundaufbau der internen Zelle eines CMOS–Gate–Arrays schematisch dargestellt; links Transistorsymbolik mit Kontakten, rechts Chiplayout. Diese Zellen sind horizontal aneinandergereiht.

Wegen der geringeren Löcherbeweglichkeit müssen die Kanäle der p–Kanal–Transistoren breiter ausgelegt sein. Jede Transistor–Source bzw. –Drain verfügt über zwei Kontaktierungen, so daß eine flexible Verschaltung möglich ist. Durch diesen Zellenaufbau sind jeweils zwei p–Kanal– bzw. zwei n–Kanal–Transistoren automa– tisch in Serie geschaltet. Die "Underpasses" werden als Verdrahtungshilfsmittel benützt.

Aus dieser Grundstruktur lassen sich auf einfache Weise logische Schaltelemente durch Festlegung der Verbindungen herstellen. Bild 3.16 zeigt einen einfachen Inverter, der aus dieser Grundstruktur erzeugt wird (siehe Bild 3.18a). Die Metallisierungsstreifen über den Source–Gebieten der beiden linken Transistoren verbinden die Kontakte mit V_{SS} (n–Kanal–Transistor) bzw. mit V_{CC} (p–Kanal–Transistoren).

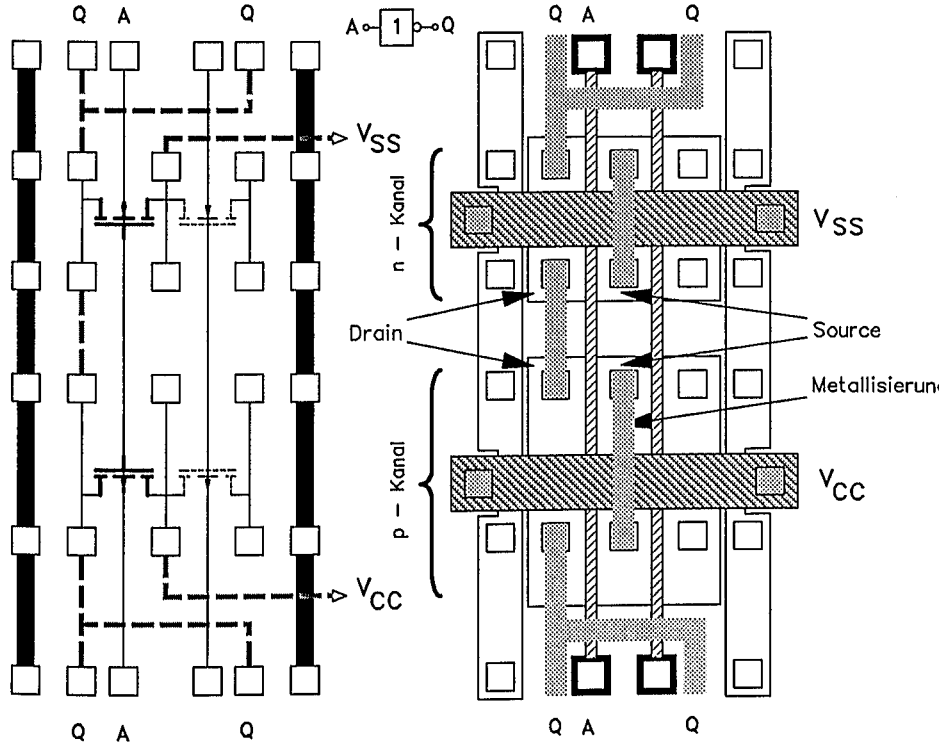


Bild 3.16: Verwirklichung eines Inverters aus dieser Grundstruktur (vgl. Bild 3.18a)

Die beiden Drain-Gebiete sind durch einen Metallisierungsstreifen (Bildmitte) verbunden. Zur Abnahme des Ausgangssignals Q stehen das obere bzw. untere Kontaktfenster des Drains zur Verfügung. Als gemeinsames Gate wird die Polysiliziumspalte (A..A) verwendet. Zwei Transistoren der Zelle bleiben ungenutzt.

Die in Bild 3.17 angegebene Numerierung der Transistoren bezieht sich auf die Numerierung im Schaltbild (siehe Bild 3.18b).

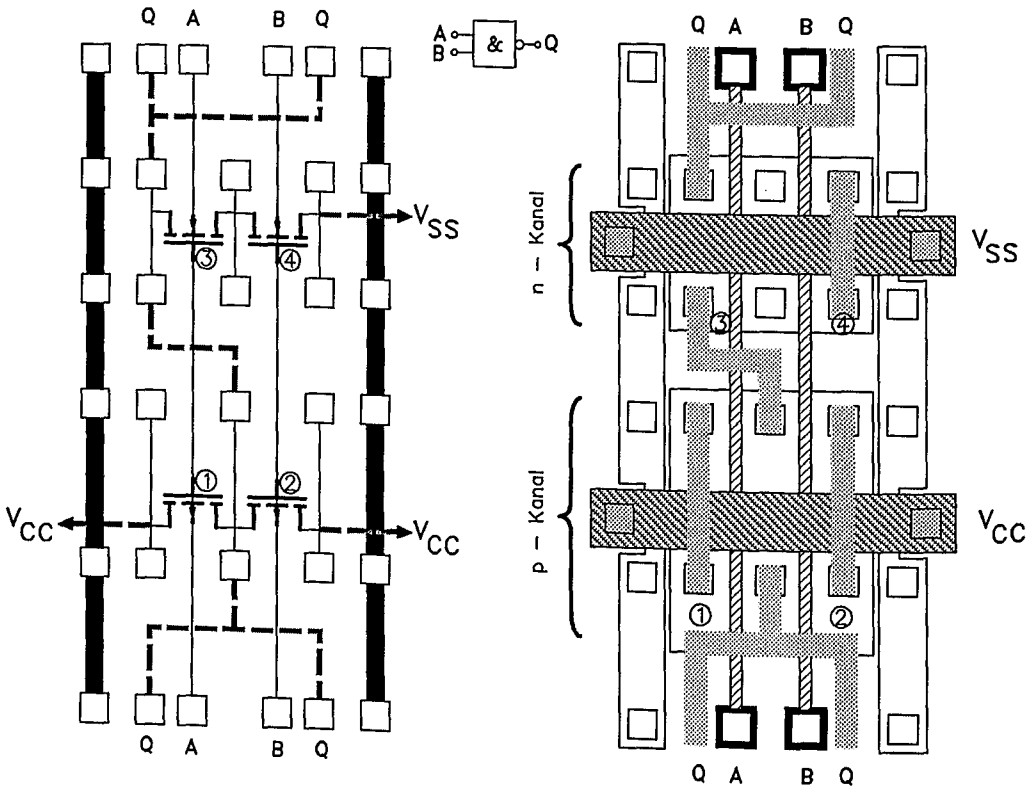


Bild 3.17: Realisierung eines 2-fach-NAND aus einer Grundzelle (vgl. Bild 3.18b)

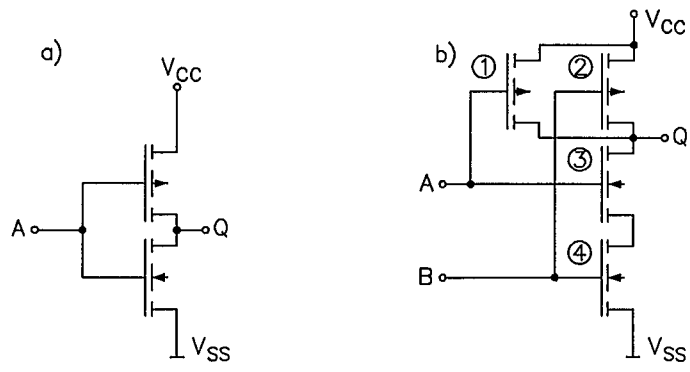


Bild 3.18: Schaltungstechnische Realisierung von a) Inverter b) 2–fach–NAND

Eine Besonderheit der CMOS–Technologie stellt die Möglichkeit dar, Schalterfunktionen in Signalpfaden zu realisieren. Als Beispiel sei das bidirektionale Transfer–Gate in Bild 3.19a erwähnt. Falls die Steuerelektrode S auf V_{CC} und $\bar{S}$ auf V_{SS} liegt, ist die Transistorstrecke A ↔ B niederohmig (siehe Bild 3.19b). Für ein bidirektionales Transfer–Gate (BXFER) werden zwei parallelgeschaltete n– und p–Kanal–Transistoren benötigt. Je nach den Spannungsverhältnissen (Logikpegeln) leitet einer der beiden Transistoren (siehe Bild 3.19c). Falls die Steuerelektrode S auf V_{SS} und $\bar{S}$ auf V_{CC} liegt, sind beide Transistoren hochohmig. In einem Schaltermodell würde dies einem geöffneten Schalter entsprechen. Transfer–Gates sind besonders nützlich für Schalter–basierende Logik–Schaltungen. (siehe auch Bild 3.26)

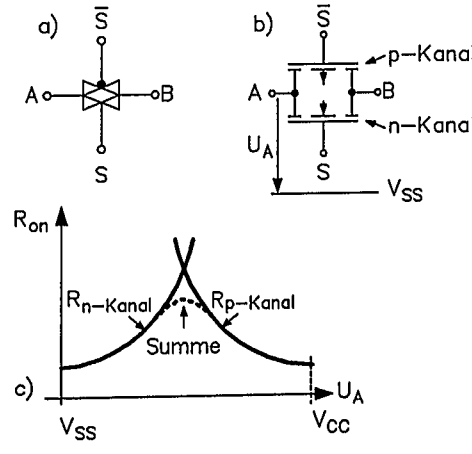


Bild 3.19: Bidirektionales Transfer–Gate, a) Logik–Schaltzeichen, b) Transistoräquivalent, c) Kanalwiderstandsdiagramm

3.3 Einflüsse der Technologie auf den Schaltungsentwurf

3.3.1 Gatterlaufzeiten

Die schaltungstechnischen Grundelemente der CMOS–Technik sind der n–FET und der p–FET. Zur Diskussion des dynamischen Verhaltens einer digitalen CMOS–Schaltung sei die Serienschaltung zweier Inverter herangezogen (siehe Bild 3.20a und b).

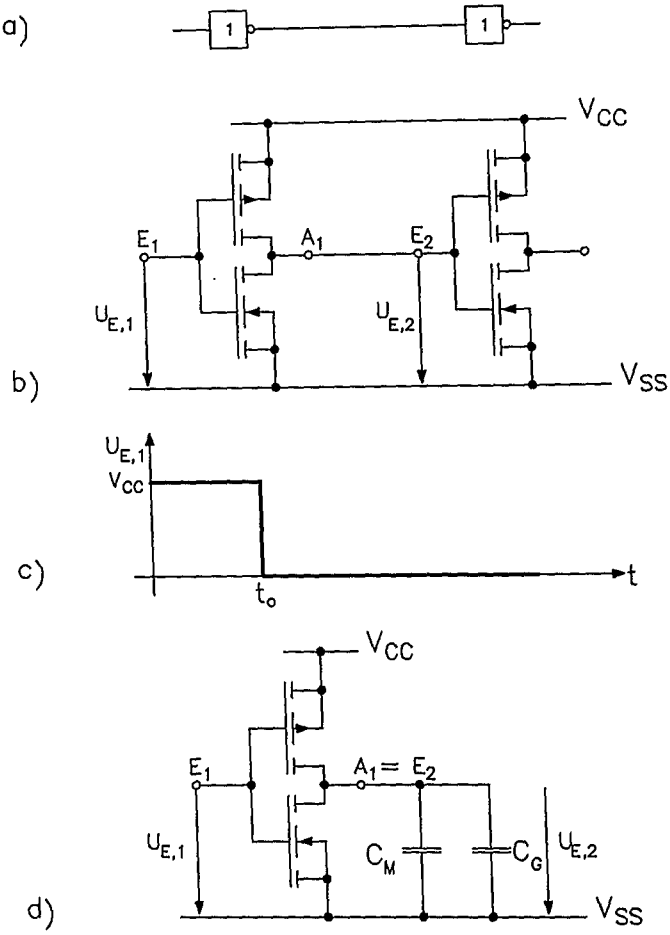


Bild 3.20: Serienschaltung zweier Inverter zur Ananlyse des Zeitverhaltens; a) Logik; b) Transistoräquivalent; c) Zeitverlauf der Eingangsspannung; d) Modell zur Ermittlung des Zeitverlaufes der Eingangsspannung des zweiten Inverters

Zum Zeitpunkt t_o soll die Eingangsspannung des ersten Inverters einen Sprung von V_{CC} nach V_{SS} machen. Die Wirkung dieses Eingangsspannungssprungs auf die Eingangsspannung des Folgeinverters kann anhand des Ersatzmodells Bild 3.20d untersucht werden. In diesem Ersatzmodell sind die Transistoren des zweiten Inverters durch deren Gate-Kapazität und die Metallverdrahtung durch deren Kapazität ersetzt worden. Der ohmsche Widerstand der Verdrahtung werde vernachlässigt. Unmittelbar vor t_o ist der p-FET gesperrt und der n-FET im ungesättigten Betriebszustand.

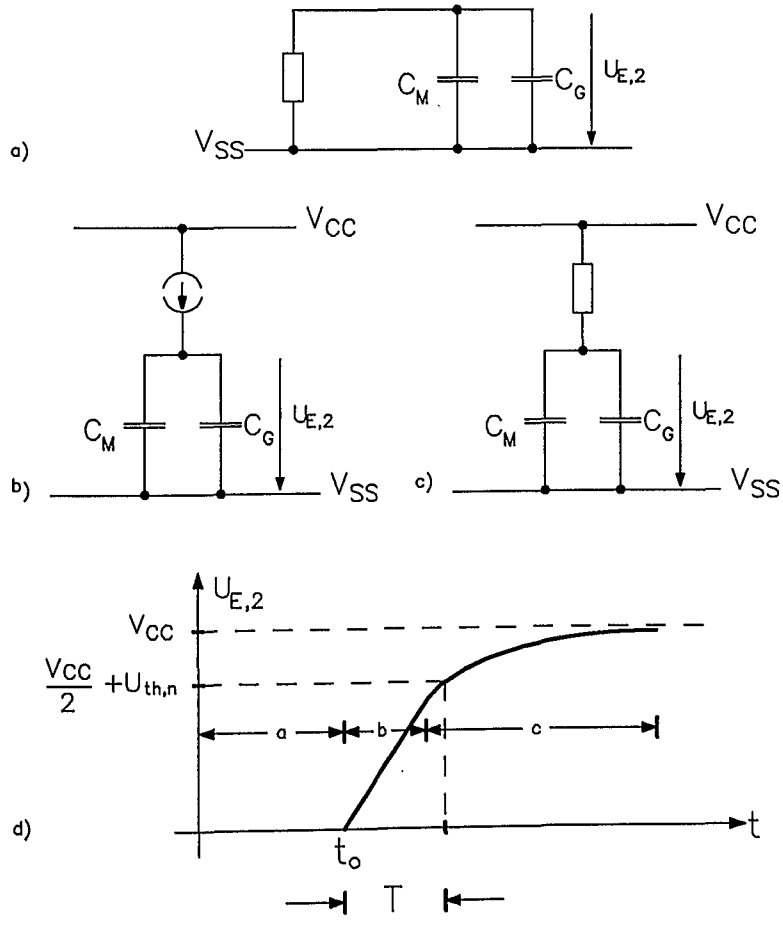


Bild 3.21: a) Ersatzschaltbild für die Schaltung in Bild 3.20b, wenn die Eingangsspannung gleich V_{CC} ist; b), c) Ersatzschaltbild für die Schaltung in Bild 3.20b zu unterschiedlichen Zeiten, nachdem die Eingangsspannung auf V_{SS} wechselt; d) Zeitverlauf der Eingangsspannung des zweiten Inverters von Bild 3.20; die Bereiche a–c beziehen sich auf die Ersatzschaltbilder in Teilbild a–c.

Die dafür gültige qualitative Ersatzschaltung ist in Bild 3.21a dargestellt. Unmittelbar nach t_0 ist der p-FET im gesättigten Zustand und der n-FET gesperrt. Der p-FET wirkt also zunächst als Stromquelle, die die Kapazitäten C_M und C_G auflädt (s. Bild 3.21b). Dadurch sinkt der Betrag der Drain-Source-Spannung des p-FET. Nachdem der Sättigungswert $|U_{DS,sat}| = |U_{GS}-U_{th,p}|$ unterschritten ist, zeigt der p-FET Widerstandsverhalten (siehe Bild 3.21c). Dieser Widerstand lädt die Metallisierungs- und Gatekapazitäten weiter auf, bis die Versorgungsspannung V_{CC} erreicht wird. Der Zeitverlauf der Eingangsspannung $U_{E,2}$ des Folgeinverters ist qualitativ in Bild 3.21d dargestellt. Erst nach einer Verzögerungszeit T erreicht die Eingangsspannung des Folgeinverters $U_{E,2}$ einen Wert, bei dem dessen Ausgangsspannung deutlich unter $V_{CC}/2$ absinkt (Bereich 4 der Übertragungskennlinie, siehe Bild 3.11c).

3.3.2 Einflußgrößen auf die Gatterlaufzeit

Die in Abschnitt 3.3.1 erläuterte Verzögerungszeit (Gatterlaufzeit) bestimmt die Schaltgeschwindigkeit. Sie wird von den folgenden Größen beeinflußt:

a) Kapazität der Verbindungsleitung (Metallisierung) zwischen dem Ausgang des Inverters und den Gatekapazitäten der Folgetransistoren: je höher die Gesamtkapazität, desto höher die Verzögerungszeit (Gatterlaufzeit) T.

b) Betriebsspannung: bei höheren Betriebsspannungen wird die Schwellenspannung, ab der der Ausgang des Folgeinverters dem Eingangssignalwechsel folgt, schneller überschritten. Die Gatterlaufzeit wird deshalb bei höheren Versorgungsspannungen kleiner.

c) Temperatur: bei höheren Temperaturen sinkt die Beweglichkeit der Ladungsträger im Kanal der Transistoren. Dadurch wird der Drainstrom der Transistoren reduziert. Zwar reduziert eine erhöhte Temperatur die Beträge der Schwellenspannungen $U_{th,n}$ bzw. $U_{th,n}$, doch die damit verbundene Erhöhung der Drainströme kann die Reduktion durch die verringerte Beweglichkeit nicht kompensieren. Die Umladung der Folgekapazitäten dauert deshalb bei höheren Temperaturen länger, d.h. die Gatterlaufzeit wird erhöht.

d) Prozeßschwankungen: durch Toleranzen bei der Herstellung der Wafer können die Transistorkennwerte Streuungen unterliegen. Beispielsweise können die Schwellenspannungen $U_{th,p}$ bzw. $U_{th,n}$ streuen. Höhere Beträge der Schwellenspannungen verringern die Drainströme. Asymmetrische Veränderungen der Schwellenspannungen können das Übertragungsverhalten positiv, aber auch negativ beeinflussen. Solche Prozeßschwankungen können deshalb die Gatterlaufzeit verkürzen, aber auch verlängern.

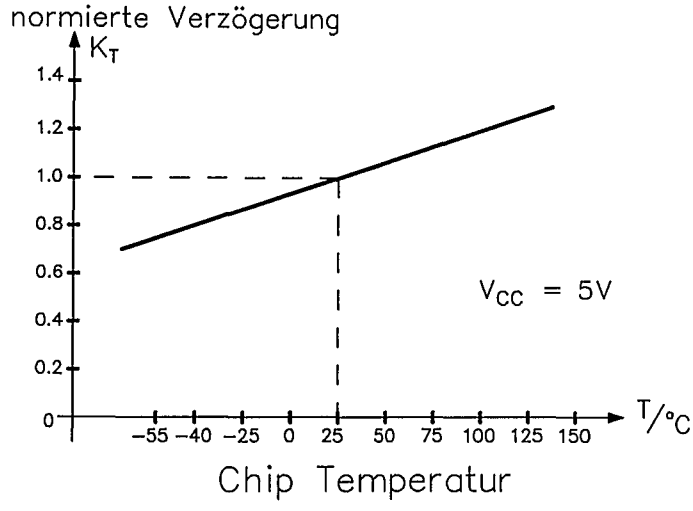


Bild 3.22: Temperaturabhängigkeit der Gatterlaufzeiten (schematisch)

Bild 3.22 zeigt eine typische Abhängigkeit der Gatterlaufzeiten in Abhängigkeit von der Temperatur bezogen auf den Wert für eine Nominalspannung V_{CC} von 5V und einer Bezugstemperatur von 25°C. Beispielsweise hat der Korrekturfaktor K_T bei −40°C den Wert 0,81 und bei 125°C den Wert 1,32. Die Spannungsabhängigkeit der Gatterlaufzeiten ist in Bild 3.23 dargestellt. Die Bezugstemperatur ist 25 °C bei einer Bezugsspannung von V_{CC} = 5V.

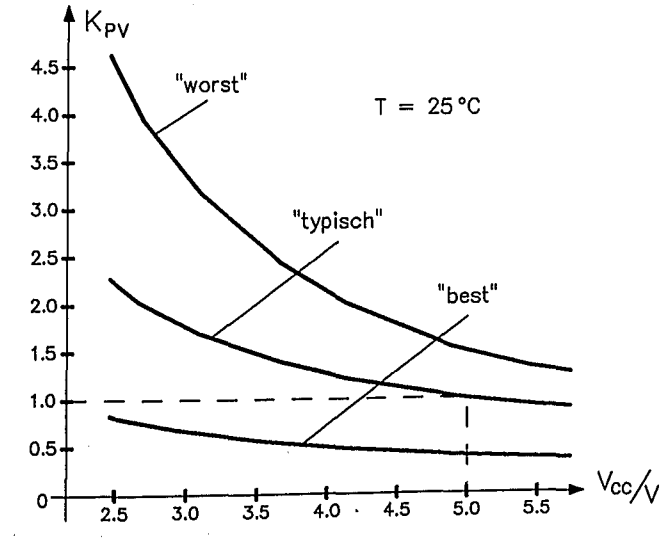


Bild 3.23: Normierte Spannungsabhängigkeit der Gatterlaufzeiten für günstigste, typische und schlechteste (worst) Verhältnisse (schematisch)

Die mittlere Kurve gilt für typische Herstellungsbedingungen. Die obere Kurve stellen die schlechtesten Prozeßbedingungen (größte Gatterlaufzeiten), die untere Kurve die besten Prozeßbedingungen (kürzeste Gatterlaufzeiten) dar. Unter Berücksichtigung aller schlechten Einflüsse, wie z.B. eine niedrige Betriebsspannung von 4,5 Volt, eine hohe Chip-Temperatur von 125°C und "worst case" Prozeßbedingungen ergibt sich ein Korrekturfaktor K von

$$K_{worst} = K_{pV} * K_T = 1{,}59 * 1{,}32 = 2{,}1$$ *(Zahlenwerte: AMS Gate-Array-Datenbuch).*

Unter Zugrundelegung aller positiven Einflüsse, wie z.B. eine Betriebsspannung von 5,5 Volt, eine niedrige Chip-Temperatur von -40^0C und "best case" Prozeßbedingungen ergibt sich ein Korrekturfaktor K von

$$K_{best} = K_{pV} * K_T = 0{,}47 * 0{,}81 = 0{,}38$$ *(Zahlenwerte: AMS Gate-Array-Datenbuch).*

Die Schwankungsbreite von Signallaufzeiten unter diesen angenommenen Betriebs- bzw. Herstellungstoleranzen beträgt damit

$$K_{worst}/K_{best} = 2{,}1 \;/\; 0{,}38 = 5{,}5 \;.$$

Schaltungen müssen deshalb so ausgelegt werden, daß sie trotz dieser großen Schwankungsbreite der Signallaufzeiten funktionsfähig sind. Präzise Vorgaben für Signallaufzeiten sind in der Regel nicht möglich.

3.3.3 Zeitabschätzungen

Gate-Array-Hersteller stellen dem Schaltungsentwickler Datenbücher zur Verfügung, in denen neben der Beschreibung der Logik-Funktion auch Angaben zur Abschätzung der Gatterlaufzeiten enthalten sind. Diese Abschätzungen der Gatterlaufzeiten basieren meistens auf der einfachen Formel

$$t_{px} = t_{dx} + K * F \;.$$

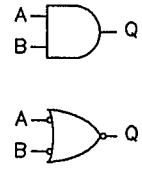
Dabei steht t_{px} für die wirksame Gatterlaufzeit t_{PLH}, wenn es sich um einen Pegelwechsel von LOW nach HIGH bzw. für t_{PHL}, wenn es sich um einen Pegelwechsel von HIGH nach LOW handelt; t_{dx} gibt die jeweilige Eigenlaufzeit t_{LH} bzw. t_{HL} des Gatters an. Das ist die Laufzeit durch ein Gatter, das durch keine nachgeschalteten Gatter belastet ist. F ist der FANOUT des Gatters. Dies ist gleichbedeutend mit der Zahl angeschlossener Gattereingänge von Folgegattern. Hierbei ist allerdings zu berücksichtigen, ob ein Eingang eines Folgegatters als einfache Last wirkt, oder ob die kapazitive Last höher ist. Deshalb wird für die Gattereingänge die Zahl der Einheitslasten angegeben, mit denen dieser Eingang ein treibendes Gatter belastet. K ist ein Maß für die Treiberstärke des treibenden Gatters, mit dem der FANOUT direkt in eine Signallaufzeit (z.B. in Nanosekunden) umrechenbar ist. Bild 3.24 zeigt eine solche Datenbuchangabe für ein einfaches Gatter.

Bei einem FANOUT von 4 errechnet sich mit den dort aufgeführten Angaben eine typische Gatterlaufzeit von 1,6 ns für einen LOW -> HIGH Übergang und von 1,4 ns für einen HIGH -> LOW Übergang. Diese Werte sind noch mit den in Abschnitt 3.3.2 erläuterten Korrekturfaktoren zu multiplizieren, wenn die Betriebstoleranzeinflüsse (Temperatur, Betriebsspannung) und die Prozeßtoleranzen berücksichtigt werden müssen.

AA02

Description
AA02 is a two-input gate, whitch performs the logical AND Funktion

Logic Symbol



Truth Table

A	B	Q
L	L	L
L	H	L
H	L	L
H	H	H

Pin Loading:

Pin with Loads	Equivalent Unit Loads
Any Input	1

Equivalent Gate Count 2
Bolt Syntax Q.AA02 AB;

Switching Charactertistics:
Conditions: V_{CC} = 5V, T_J = 25 °C, Typical Process

Max. Delay (ns) Form	T_o	Parameter	Number of Units Loads (F) 1	2	3	4	8	Instrinsic Parameters t_{dx}	K
Any Input	Q	t_{PLH}	1.1	1.3	1.4	1.6	2.3	0.9	0.18
		t_{PHL}	1.0	1.1	1.3	1.4	1.8	0.9	0.12

Propagation Delay Equation: $t_{px} = t_{dx} + K * F$

Bild 3.24: Beispiel einer Datenbuchbeschreibung einer Makrozelle (AMS)

3.3.4 Setup-/Hold-Zeiten

Wenn ein flankengetriggertes Flipflop Daten übernehmen soll, muß sichergestellt sein, daß der Logik-Pegel am Dateneingang eine gewisse Vorbereitungszeit (setup-Zeit, t_{SU}) vor der aktiven Taktflanke und noch eine gewisse Nachhaltezeit (hold-Zeit, t_H) nach dieser Taktflanke stabil ist (siehe Bild 3.25). Das Berechnungsverfahren zur Bestimmung dieser Zeiten kann Bild 3.26 entnommen werden. Dort ist der typische Aufbau eines MASTER-SLAVE-D-Flipflops dargestellt. Es sei angenommen, daß der Takteingang auf LOW liegt. Dadurch ist das Transfer-Gate G4 geöffnet. Die Transfer-Gates G3 und G8 sind gesperrt. Das Daten-Signal durchläuft also die Gatter G4, G5, G6 und steht am Gatter G3 an.

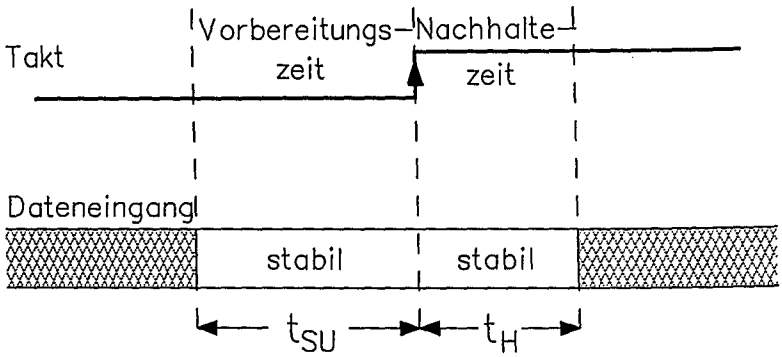


Bild 3.25: Definition von Vorbereitungs- und Nachhaltezeit

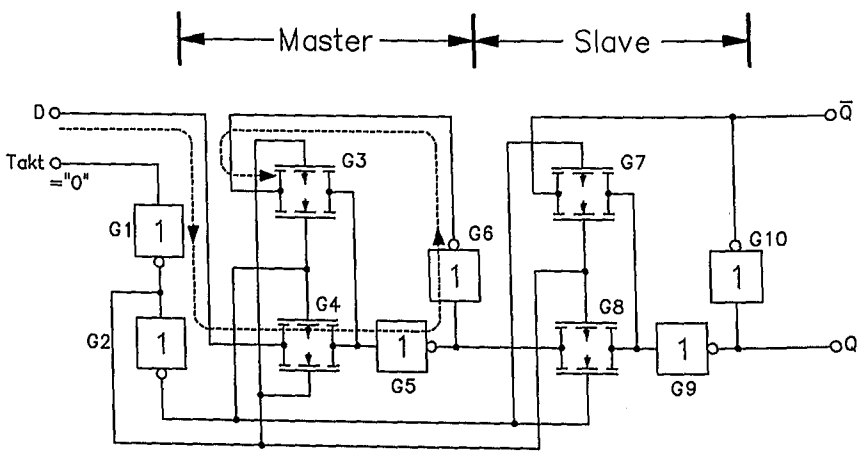


Bild 3.26: Modell eines D-FF zur Ermittlung der charakteristischen Zeitparameter

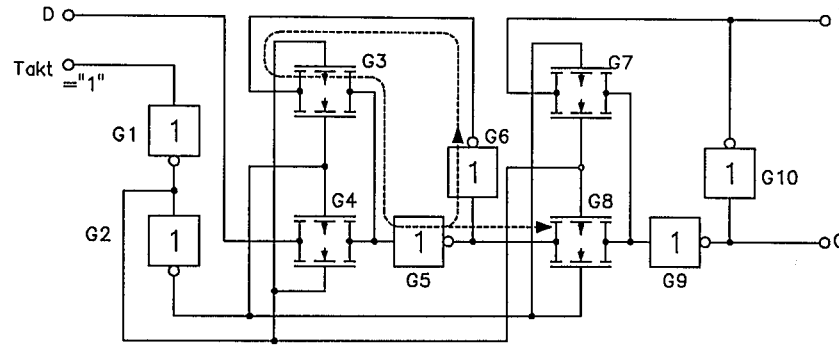


Bild 3.27: Modell eines D–FF zur Ermittlung der charakteristischen Zeitparameter

Wenn das Taktsignal auf HIGH wechselt, werden die Transfer–Gates G3, G8 geöffnet und G4 wird geschlossen. Das am Eingang von G3 anstehende Signal wird zum Slave–Flipflop weitergereicht (siehe Bild 3.27).
Da G4 gesperrt ist, werden Änderungen des Datensignals am Eingang abgehalten. Die beschriebene Funktion ist nur gewährleistet, wenn das Signal am Eingang von G3 zu dem Zeitpunkt bereits stabil ist, zu dem das Transfer–Gate geöffnet wird. Diese Vorbereitungs–Zeit (setup–Zeit) berechnet sich zu

$$t_{su} = (t_{PD,G4} + t_{PD,G5} + t_{PD,G6}) - (t_{PD,G1} + t_{PD,G2})$$

wobei $t_{PD,Gi}$ die Signallaufzeiten durch die beteiligten Gatter sind. Andererseits darf sich das Daten–Eingangssignal nicht ändern, bevor das Transfer–Gate G4 geschlossen ist. Die Hold–Zeit ist deshalb

$$t_H = t_{PD,G1} + t_{PD,G2} \; .$$

Üblicherweise werden die Zeiten t_{su} und t_H in den Makrozell–Datenbüchern spezifiziert. Beispielsweise gibt AMS für das D–Flipflop DF08 eine Setup–Zeit von 1,6 ns und die Hold–Zeit mit 0,6 ns an. Werden diese Zeiten unterschritten, so ist die Funktion des Flipflops nicht garantiert. Es kann dabei in einen metastabilen Zustand eintreten, in dem die Ausgänge keinen gültigen Logik–Pegel annehmen. Dieser Zustand kann vergleichsweise lang andauern. Es ist auch möglich, daß die Ausgangssignale des Flipflops oszillieren.

Zusätzlich zu den Setup– bzw. Hold–Zeiten muß eine Minimaldauer eingehalten werden, die das Takt–Signal LOW bzw. HIGH sein muß (minimum pulse width). Die minimale LOW–Zeit wird durch die Laufzeit durch die Gatter G4, G5 und G6 bestimmt:

$$t_{PW,low} = t_{PD,G4} + t_{PD,G5} + t_{PD,G6} \; .$$

Falls diese Zeit unterschritten wird, steht für das Daten-Eingangssignal nicht genügend Zeit zur Verfügung, um zum Transfer-Gate G3 durchzukommen. Die minimale HIGH-Zeit ist durch die Laufzeit der Gatter G8, G9 und G10 bestimmt:

$$t_{PW,high} = t_{PD,G8} + t_{PD,G9} + t_{PD,G10} .$$

Ansonsten erhält das im Master-Flipflop zwischengespeicherte Signal nicht genügend Zeit, um das Transfer-Gate G7 zu erreichen. Da das Slave-Flipflop Nachfolge-Gatter treibt, können sich diese Zeiten FANOUT-abhängig verlängern.

3.3.5 Maximale Kippfrequenz / maximale Taktfrequenz

Die Geschwindigkeit von Gate-Arrays wird häufig durch eine "maximum toggle frequency" charakterisiert. Damit ist die maximale Kippfrequenz eines einfachen Flipflops ohne weitere Beschaltung gemeint. Beispielsweise gibt Texas Instruments für ihre 1 µ CMOS Gate-Array-Familie TGC100 maximale Kippfrequenzen von 95 bis 208 MHz an. Diese Frequenzen sind wesentlich höher, als die schaltungstechnisch nutzbaren Taktfrequenzen. Dies kann anhand von Bild 3.28 erläutert werden.

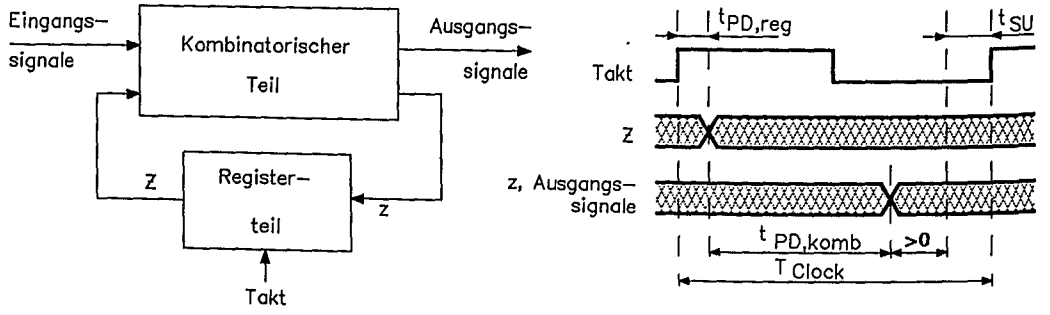


Bild 3.28: Zur Ermittlung der maximalen Taktfrequenz

Danach besteht eine digitale Schaltung aus kombinatorischen Schaltungsteilen und speichernden Schaltungsteilen, die mit Flipflops oder Registern aufgebaut werden. Nach einer aktiven Taktflanke gibt der Registerteil mit einer Verzögerungszeit $t_{PD,Reg}$ neue Werte aus, die mit einer Durchlaufzeit $t_{PD,Komb}$ durch die Kombinatorik wieder an den Registereingängen angelegt werden. Die neuen Eingangsdaten der Register müssen spätestens die Vorbereitungszeit t_{SU} vor der Folgetaktflanke stabil sein. Die Taktperiode T_{Clock} muß deshalb die Bedingung

$$T_{Clock} > t_{PD,Reg} + t_{PD,Komb} + t_{SU}$$

unter Berücksichtigung der "worst case"-Verhältnisse erfüllen. Dazu ist der längste Signalpfad durch die Logik und dessen Signallaufzeit zu ermitteln.

3.4 Entwurfsgrundsätze

Synchrones Design: Die Laufzeiten von Signalen auf einem Gate Array unterliegen insbesondere bei automatischer Plazierung und Verdrahtung Toleranzen. Deshalb ist die reale Laufzeit zum Zeitpunkt der Logikentwicklung nicht präzise bekannt. Die Schaltungsfunktion darf deshalb nicht von präzisen Signallaufzeiten abhängen. Deshalb muß asynchrone Logik zugunsten synchroner Logik vermieden werden.

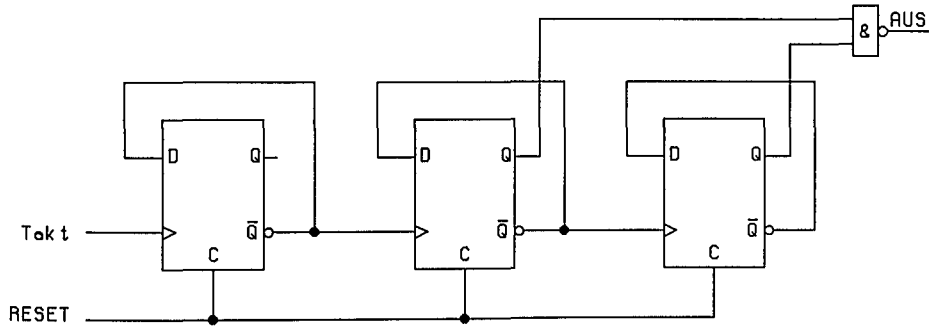


Bild 3.29: Asynchroner Ausgang eines Zählers

Synchrone Logik beinhaltet Signalpfade durch die Logik, deren Ausgangszeitverhalten nicht von den Laufzeiten der Logikelemente im jeweiligen Logikpfad abhängt. Jeder Wechsel des Ausgangslogikpegels der Schaltung wird durch einen Taktimpuls ausgelöst. Die minimale Taktperiode wird durch die maximale Verzögerungszeit durch den Logikpfad bestimmt. Der asynchrone Ausgang des Zählers in Bild 3.29 kann Spikes ausgeben. In Bild 3.30 ist das Ausgangssignal synchronisiert, falls die Taktperiode größer ist, als die Laufzeit durch ein Flipflop, das NAND-Gatter und die Setup-Zeit des Flipflops 4.

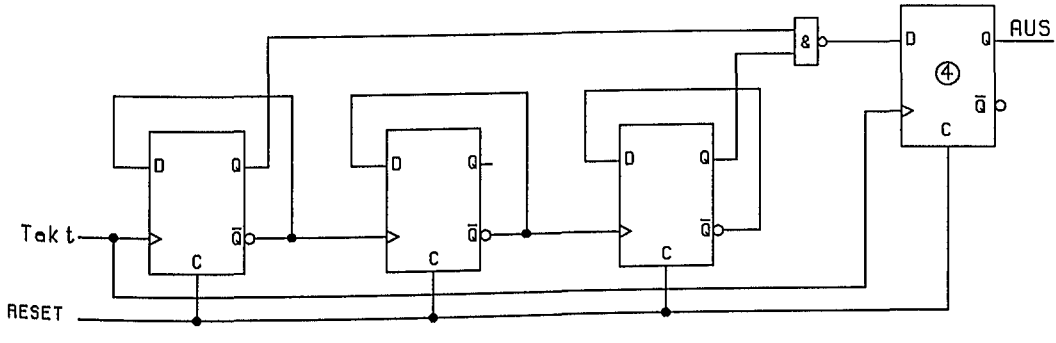


Bild 3.30: Synchroner Ausgang eines Zählers

Vermeidung von Spike–behafteten Schaltungen: Die Eingänge des Decoders von Bild 3.31 treiben mehrere interne Gatter. Ein oder mehrere Ausgänge können Spikes bzw. Hazards erzeugen, wenn sich der Logikpegel eines Eingangs ändert. Durch Verwendung eines synchronen Strobe–Signals kann der Ausgang spikefrei gemacht werden (Bild 3.32).

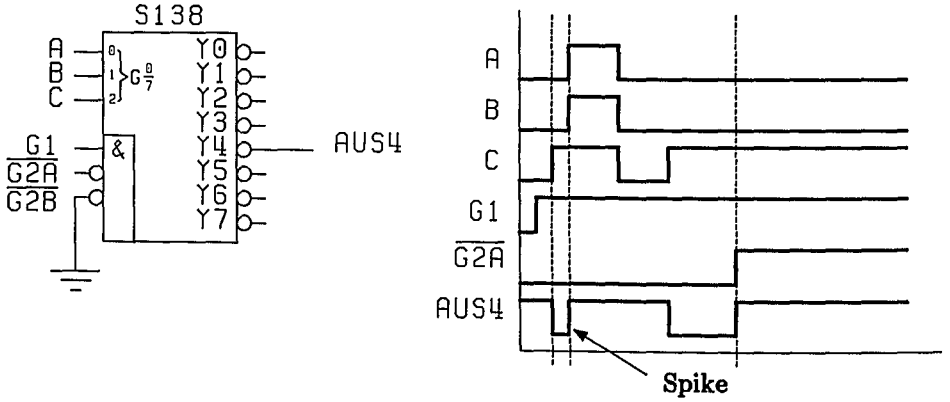


Bild 3.31: Asynchrone Decodierung erzeugt Spikes

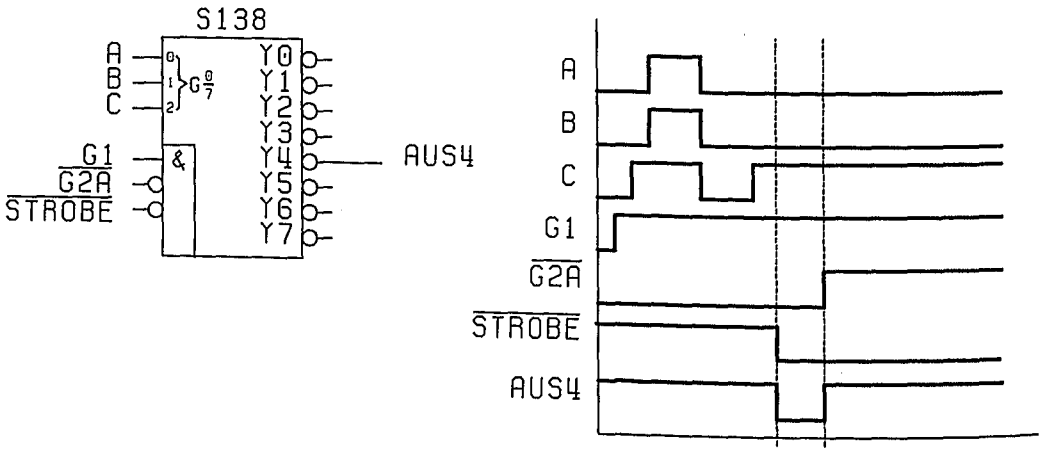


Bild 3.32: Synchrone Decodierung gewährt Spike–Freiheit

Vermeidung "gegateter" Clocks: Hazards (manchmal auch Spikes genannt) entstehen, wenn sich die Eingangssignale eines Gatters gleichzeitig in entgegengesetzter Richtung ändern (Bild 3.33). Wenn ein Gatter dazu benutzt wird, den Takt für ein Flipflop freizugeben bzw. zu sperren, können nicht vorhersagbare Clock–Spikes auftreten (Bild 3.34a). Statt den Takt des Flipflops zu steuern, ist es besser, über einen Multiplexer den D–Eingang zu kontrollieren (Bild 3.34b).

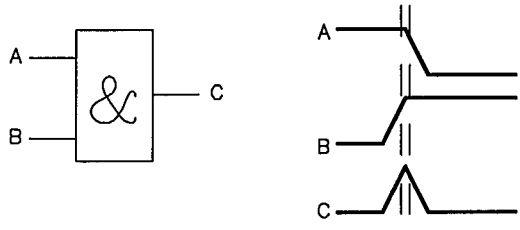


Bild 3.33: Gleichzeitige Änderung der Eingangssignale in entgegengesetzter Richtung erzeugen Spikes

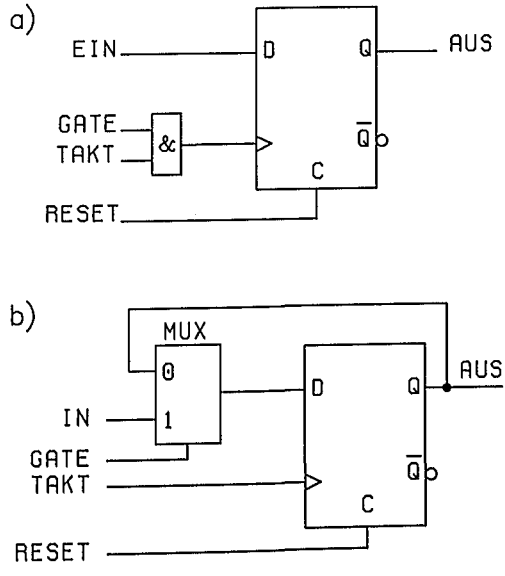


Bild 3.34: a) Gefahr falscher Taktimpulse durch Clock–Spikes
b) Simulation des "Takt–Enable" durch Flipflop–Rückkopplungen über einen Multiplexer

Vermeidung von Wettrennen: Wettläufe von zwei Signalen (race) können entstehen, wenn sie auf unterschiedlich langen Wegen zum selben Gatter laufen (Bild 3.35).

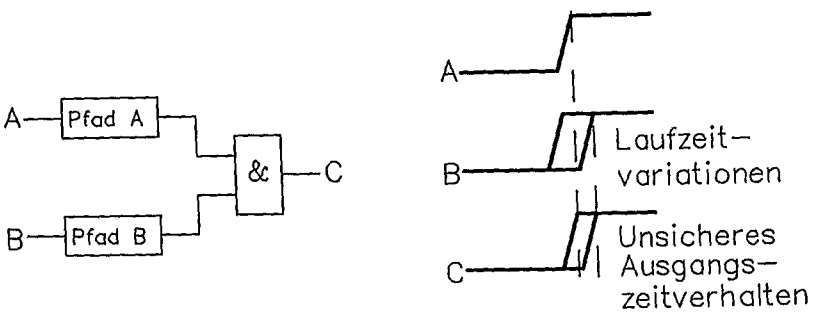


Bild 3.35: Laufzeitvariationen erzeugen unsicheres Verhalten einer Logikschaltung

Vermeidung von Rückkopplungen: Bild 3.36 zeigt zwischen zwei Flipflops einen rückgekoppelten Logik-Pfad. Dessen Funktion hängt von den Laufzeitverzögerungen des Rückkopplungspfades ab und kann je nach Verdrahtungslängen zu langsam sein, um einlaufende Signale abzuschalten.

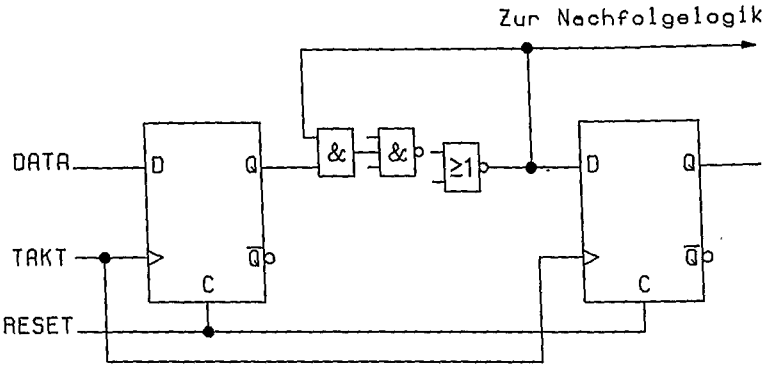


Bild 3.36: Unsichere Funktion durch asynchrone Rückkopplung

Begrenzung der Belastung interner Logik: CMOS kann im Prinzip eine beliebige Zahl von Gattern treiben. Es wird lediglich die Signallaufzeit erhöht. Gatterbelastungen mit mehr als etwa 1 pF sollten deshalb und auch wegen der langen Anstiegs-/Abfallzeiten vermieden werden. Lange Anstiegs-/Abfallzeiten gefährden die korrekte Funktion von Flipflops.

Externe Flipflop-Kontrolle: Alle Flipflops, Register und Latches müssen über die Gehäuseanschlüsse in einen bekannten Zustand gesetzt werden können. Zur Vereinfachung des Tests sollte ein Master-Reset vorgesehen werden.

Anschluß unbeschalteter Eingänge: Unbenutzte Eingänge von Logik–Zellen müssen mit den Logik–Level–Klemm–Zellen beschaltet werden.

Kurzgeschlossene Makrozelleneingänge: Logisch äquivalente Eingänge einer Makrozelle dürfen in der Regel nicht kurzgeschlossen werden. Statt dessen müssen unbenutzte Eingänge mit Hilfe entsprechender Makrozellen auf "1" bzw. "0" "geklemmt" werden.

Interne Busse: Busse werden durch einen oder mehrere 3–State–Treiber getrieben. Es dürfen nicht alle Treiber gleichzeitig abgeschaltet sein. Andererseits darf nie mehr als ein Treiber gleichzeitig aktiviert sein, weil sonst die Gefahr eines Kurzschlusses zwischen V_{CC} und V_{SS} besteht. Wenn möglich, Busse vermeiden und statt dessen Multiplexer verwenden.

3.5 Testprobleme integrierter Schaltungen

3.5.1 Einführung

Der Fortschritt der Halbleitertechnologie erlaubt die Herstellung von digitalen Schaltungen mit einer sehr großen Zahl von Transistoren bzw. Gattern pro Chip. Diese Komplexität ist mit einer gewaltigen Zunahme des Verhältnisses der Gatteranzahl zur Anzahl der Ein-/ Ausgangsanschlüsse gekoppelt. Dies verschlechtert die Kontrollierbareit und Beobachtbarkeit der Logik auf dem Chip. Diese Situation verursacht bereits Probleme beim Testen eines einzelnen Chips. Wenn man ein ganzes System betrachtet, das aus vielen Platinen besteht, die jeweils mit vielen Chips bestückt sind, wird die Testausgabe überwältigend.

In digitalen Systemen müssen Tests durchgeführt werden, um Fehler der Fertigung oder durch Verschleiß entstandene Fehler zu erkennen. Tests werden dabei auf verschiedenen Ebenen durchgeführt: die Siliziumplättchen (Dies) werden während der Fertigung geprüft, die im Gehäuse verpackten Chips vor der Platinenbestückung, die Platinen vor der Systemzusammenstellung und das System nach der Fertigstellung.

3.5.2 Physikalische Fehler und Fehlermodelle

Bei früheren Small Scale IC (SSI) waren Bonddraht-Fehler die häufigsten Fehlerursachen. Da das Zahlenverhältnis zwischen Pins und Logik relativ groß war, konzentrierten sich Tests auf die Detektierung von Fehlern der I/O-Verbindungen. VLSI-Systeme weisen dagegen wesentlich komplexere Fehlerarten auf. Unterschieden werden Herstellungsfehler und Fehler während des Einsatzes durch Verschleiß. Beide Fehlerklassen sind aber nicht streng disjunkt, da herstellungsbedingte Fehlerursachen erst im Betrieb zum Ausfall führen können.

Fertigungsbedingte Fehler sind beispielsweise fehlerhafte Einzeltransistoren, Unterbrechungen in Verbindungsleitungen auf verschiedenen Ebenen (Polysilizium, Diffusionsbereiche, Metallisierung ...), Kurzschlüsse zwischen und innerhalb der verschiedenen Chipebenen. Defekte in der Kristallstruktur führen zu Fehlern in der Umgebung des Kristallbaufehlers. Wegen der extrem kleinen Geometrien und der damit verbundenen engen Toleranzen, können Justage- oder Maskenfehler und Lithographiefehler zu Löchern im Oxid, fehlerhaften Kontakten, fehlerhaften Transistoren (fehlende oder überzählige) oder zu Kurzschlüssen führen. Ungenaue Dotierungsprofile können Transistoren mit unerwünschten Kennwerten ergeben, die zu zeitweisen oder andauernden Chipausfällen führen. Unsachgemäße Handhabung kann insbesondere bei MOS-Schaltungen zu Gate-Durchschlägen führen. Schlechte Verkapselung kann zum Eindringen von Feuchtigkeit ins Gehäuse führen. Dies hat Langzeitfehler durch Korrosion zur Folge. Auch können die unterschiedlichen Ausdehnungskoeffizienten zu Brüchen im Substrat oder in Verbindungen führen. Radioaktive Verunreinigungen im Gehäusematerial können durch niederenergetische Strahlung elektrische Speicherladungen löschen.

Zu den Langzeitfehlermechanismen gehören Leitungsbrüche und -kurzschlüsse, Degradation und Durchbruch aktiver Elemente. Die Aluminiummetallisierung kann Korrosion ausgesetzt sein. Hohe Stromdichten können in dünnen Drähten zu Metallwanderung und damit eventuell zu einer Unterbrechung führen. Die Ausbildung von Kristallauswüchsen (Whisker) kann zu Widerstandsbrücken führen. Die Wanderung von Alkaliionen kann die Schwellenspannungen einzelner Transistoren verändern. Hohe elektrische Feldstärken können "heiße" Elektronen erzeugen, die ins Gateoxid von MOS-Transistoren eindringen, dort steckenbleiben und so die Schwellenspannung verändern. Dadurch können vorübergehende aber auch permanente Fehler entstehen.

Diese noch unvollständige Auflistung einiger Fehlermechanismen macht bereits deutlich, daß wegen der Vielzahl und komplexen Natur der physikalischen Fehler aus praktischen Gründen ein Chiptest nicht unmittelbar mit physikalischen Fehlern arbeiten kann. Vielmehr beschränkt man sich auf die Feststellung, ob überhaupt ein Fehler vorliegt, ohne sich um die genaue Fehlerart zu kümmern. Dabei wird angenommen, daß ein physikalischer Fehler das Verhalten einer Schaltung verändert und dieses Fehlverhalten durch eine Testsignalfolge über abweichende Ausgangssignale feststellbar ist. Unterschiedliche physikalische Fehlerursachen können dabei dasselbe Fehlverhalten aufweisen.

Die Zusammenfassung der möglichen physikalischen Fehler erfolgt durch Fehlermodelle. Die Fehlermodelle beschreiben dabei die möglichen Effekte physikalischer Fehler auf abstrakteren Ebenen, wie z.B. der Gatterebene, der Registertransfer-Ebene oder der Ebene der funktionellen Blöcke. Wenn ein Fehlermodell die interessierenden physikalischen Fehlermöglichkeiten genügend genau erfaßt, genügt es, den Test auf der entsprechenden abstrakteren Ebene durchzuführen, um das Fehlen oder Vorhandensein irgendwelcher physikalischer Fehler feststellen zu können.

3.5.2.1 Fehlermodell auf Gatterebene

Auf Gatterebene wird sehr oft das stuck-at-Fehlermodell angewandt. Dabei wird angenommen, daß der Effekt der physikalischen Fehlerursachen sich so äußert, als ob einzelne Ein- bzw. Ausgänge logischer Gatter beständig den logischen Pegel "1" (stuck-at 1) oder "0" (stuck-at 0) annehmen. Als Beispiel sei der einfache CMOS-Inverter von Bild 3.37 betrachtet. Durch eine logische "1" am Eingang X wird der n-Kanal-Transistor leitend, bzw. im Schaltermodell betrachtet, geschlossen, und der Ausgang Z nimmt den logischen Pegel "0" ein. Der p-Kanal-Transistor bleibt gesperrt, bzw. im Schaltermodell betrachtet, geöffnet. Bei einer logischen "1" am Eingang kehren sich die Verhältnisse um. Falls durch einen Fehler der Eingang X mit Masse kurzgeschlossen ist, bleibt der Ausgang beständig auf "1". Dasselbe gilt, falls bei A eine Unterbrechung vorliegt, da nach einer gewissen Zeit die Ladungen von beiden Gates über die Isolationswiderstände abgeflossen sein werden. Falls aber beispielsweise die Unterbrechung bei B erfolgt, ergibt eine logische "0" am Eingang am Ausgang eine (korrekte) logische "1". Wird jedoch am Eingang eine logische "1"

angelegt, sind beide Transistoren gesperrt. Der Ausgang Z behält für eine gewisse Zeit seinen vorherigen Wert bei, bis durch Leckströme eine Umladung erfolgt. Diese Umladungszeit kann in der Größenordnung von einigen ms liegen. Falls unter Zugrundelegung üblicher Arbeitsfrequenzen der Invertereingang zwischen "0" und "1" wechselt, äußert sich die Unterbrechung B wie ein stuck-at-1 Fehler am Ausgang Z. Falls durch einen Fehler der n-Kanal-Transistor permanent leitend ist, äußert sich eine logische "1" am Eingang durch eine Spannung am Ausgang Z, die zwischen Masse und Vcc liegt und deren Wert vom Widerstandsverhältnis der beiden Transistoren abhängt.

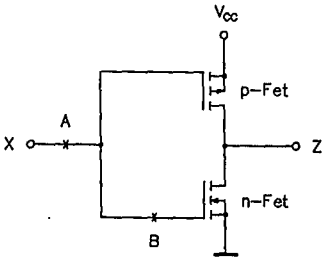


Bild 3.37: Zur Erläuterung von Fehlermodellen

Eine weitere Fehlerklasse sei anhand des NOR-Gatters von Bild 3.38 erläutert, Dazu sei angenommen, daß bei A eine Unterbrechung vorliegt. Normalerweise sollte der Ausgang Z bei X=1 und Y=0 den Wert 0 annehmen. Durch den Fehler existiert jedoch kein leitender Pfad von Z zu Vcc bzw. Masse. Z behält also seinen bisherigen Wert bei. Der Ausgang kann jedoch durch Y=1 auf "0" bzw. durch X=Y=0 auf "1" gezwungen werden. Das NOR-Gatter "erinnert" sich manchmal durch den Fehler bei A an frühere Werte und wird dadurch sequentiell.

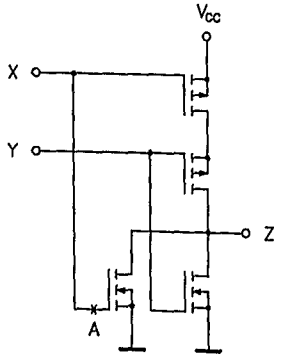


Bild 3.38: Ein NOR-Gatter mit Unterbrechung bei A wird "sequentiell"

3.5.3 Elementare Testkonzepte

Das Grundproblem des Testens besteht in der Ermittlung einer optimalen Testvorschrift für eine gewisse Schaltung S unter Zugrundelegung einer Menge **F** von Fehlern. Die Fehlermenge soll dabei alle Fehler modellieren, die voraussichtlich in der Schaltung auftreten können. Zur Testvorschrift gehören drei wesentliche Schritte: 1. die Testgenerierung, 2. die Testdurchführung und 3. die Testverifikation.

Die Testgenerierung umfaßt die Entwicklung eines Testdatensatzes bestehend aus einer Menge von Test-Vektoren **T**(F). Die zu testende Schaltung wird mit den Testfolgen des Testdatensatzes beaufschlagt. Falls die Schaltung Fehler enthält, ergeben sich an einem oder mehreren Ausgängen der Schaltung S fehlerhafte Logiksignale. **T**(F) ist dann ein vollständiger Testdatensatz, wenn er für jeden Fehler f in der Fehlermenge **F** mindestens einen Test enthält, der f erkennt.

Nachdem ein Testdatensatz entwickelt wurde, kann dieser mit einem Testautomaten an die Eingänge der Schaltung (primary inputs) in Form elektrischer Pegel angelegt und die Schaltungsreaktion an deren Ausgängen (primary outputs) unter Einhaltung vorgegebener Zeitbeschränkungen gemessen werden.

Nach der Testdurchführung und Erfassung der Reaktion der Schaltung auf den Testdatensatz müssen die gewonnenen Signale ausgewertet und auf das Vorhandensein von Fehlern in der Schaltung überprüft werden (fault detection). Schließlich kann als Teil der Testverifikation eine Fehlerlokalisierung bzw. Fehlerdiagnose in der Schaltung S vorgenommen werden. Die Realisierung von Fehlererkennung (fault-detection) ist relativ einfach. Sie liefert jedoch nicht unmittelbar eine Aussage über die Fehlerursache. Fault-Detection wird oft als Wafer-Probe-Test durchgeführt, um gute Schaltungen von schlechten Schaltungen vor der Verpackung in Gehäusen zu trennen. Fehlerlokalisierung ist wichtig bei der Qualifikation und Charakterisierung von Prototypen eines neuen Chips.

Drei Grundkonzepte sind in diesem Zusammenhang wesentlich: die Sensibilisierung, die Konsistenz und die Untestbarkeit. Diese können anhand der kombinatorischen Schaltung in Bild 3.39 unter Zugrundelegung des Stuck-at-Fehlermodells erläutert werden. Die Schaltungseingänge (primary inputs) sind mit $X_1..X_6$, der Schaltungsausgang (primary output) ist mit Z bezeichnet. Alle Netze der Schaltung sind numeriert. Verzweigende Netze tragen an allen Netzenden eigene Netznummern, da jedes Netzende individuelle Fehler enthalten kann. Ein Stuck-at-1-Fehler am Netzende 11 des Netzes 8-10-11 werde als 11/1, ein Stuck-at-0-Fehler an derselben Stelle als 11/0 gekennzeichnet. Die Schaltung in Bild 3.39 solle den Fehler f=7/0 enthalten. Um diesen Fehler testen zu können, muß ein Testvektor einen Signalwechsel des Netzes 7 auslösen und gleichzeitig sicherstellen, daß die Auswirkungen dieses Wechsels am Ausgang Z (Netz 15) beobachtbar ist. Dies kann durch einen Testvektor erreicht werden, der Netz 7 auf "1" setzt und den Ausgang (Netz 15) auf Netz 7 sensibilisiert. Die Verfolgung des Signalwegs von 7 nach 15 zeigt, daß Netz 10 auf "0" und Netz 14 auf "1" kontrolliert werden muß. Andernfalls wäre der Ausgangs-

pegel bei Z nicht durch den Pegel von Netz 7 bestimmt, d.h. der Ausgang wäre für Netz 7 desensibilisiert. Bei der Sensibilisierung muß auf mehrfache Signalwege vom Fehlerort zum Beobachtungspunkt geachtet werden. Der Fehler f=8/0 in Bild 3.39 kann über die Pfade (8,10,13,15) bzw. (8,11,12,14,15) am Ausgang sensibilisiert werden. Falls jedoch beide Pfade gleichzeitig sensibilisiert wären, würden sich die Signale der beiden Pfade am Gatter UND4 gegenseitig blockieren. Gleichzeitige Sensibilisierung mehrerer Pfade kann offensichtlich dazu führen, daß effektiv keine Sensibilisierung vorliegt, und der Fehler nicht detektierbar ist.

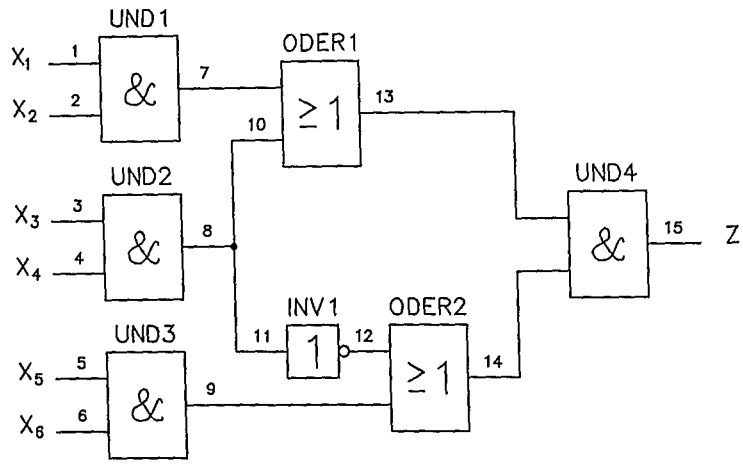


Bild 3.39: Schaltung zur Erläuterung der Testkonzepte "Sensibilisierung, Konsistenz, Untestbarkeit"

Der erste Schritt zur Entwicklung eines Testvektors ist die Formulierung der Bedingungen zur Erzeugung eines Signalwechsels an einer bestimmten Stelle der Schaltung (z.B. Netz 7 auf "1"; s.o.) und der Sensibilisierung eines Beobachtungspfads. Im zweiten Schritt muß analysiert werden, ob und welche Testvektoren existieren, die diese Bedingungen gleichzeitig erfüllen. Diese Konsistenzanalyse erfordert die Rückwärtsuntersuchung der Schaltung bis zu den Eingängen. Für den oben diskutierten Fehler f=7/0 wird mit (X_1=1;X_2=1) die Testbedingung gesetzt und mit (X_3=0;X_4=x;X_5=x;X_6=x) bzw. (X_3=x;X_4=0; X_5=x;X_6=x) der Beobachtungspfad sensibilisiert (x="0" oder "1"; d.h. "Don't Care"). Also sind die Bitmuster (Testvektoren) (110xxx) und (11x0xx) die Menge der Tests **T**(7/0), mit denen der Fehler 7/0 in der Schaltung in Bild 3.39 feststellbar ist. Im allgemeinen ist die Konsistenzprüfung sehr aufwendig, da einzelne Sensibilisierungsbedingungen zu einander widersprechenden Forderungen führen können.

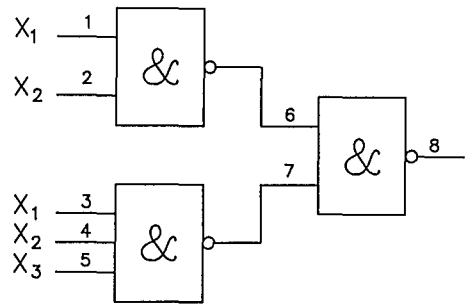


Bild 3.40: Untestbare redundante Schaltung; der Fehler 5/1 ist untestbar.

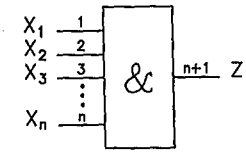
Falls für einen Fehler f einer Schaltung S die Menge der Testvektoren **T**(f) leer ist, ist der Fehler untestbar, d.h. es gibt keine Möglichkeit im Test festzustellen, ob der Fehler f in der Schaltung vorliegt oder nicht. Das zu f gehörende Netz ist also ein redundantes Netz. Ein einfaches redundantes kombinatorisches Netz ist in Bild 3.40 dargestellt. Der Fehler f=5/1 ist untestbar, da zur Sensibilisierung die Netze 3,4 und 6 auf "1" gesetzt werden müssen. Die Konsistenzprüfung führt dabei auf den Widerspruch $X_1=X_2=1$ und $X_1 \& X_2=0$. Die Schaltung verhält sich offensichtlich im fehlerfreien Fall und mit dem Fehler f=5/1 gleich. Die Schlußfolgerung, dieser Fehler sei harmlos und kann ignoriert werden, ist jedoch nicht erlaubt. Beispielsweise ist der Vektor (110) ein Test für den Fehler f=1/0 der Schaltung in Bild 3.40. Falls jedoch der nicht detektierbare Fehler f=5/1 ebenfalls in der Schaltung enthalten ist, ist f=1/0 durch (110) nicht feststellbar. Die Schaltung wird also durch den Test als fehlerfrei charakterisiert, obwohl der Doppelfehler (1/0;5/1) enthalten ist. Nicht testbare Fehler können also Testaussagen für andere Fehler ungültig machen, falls beide gleichzeitig in der Schaltung enthalten sind.

3.5.4 Testgenerierung

Die vorstehenden Ausführungen erläutern einige Aspekte, die Stuck-at-Fault- Testing zu einer komplexen Aufgabe machen. Für gewisse Klassen kombinatorischer Schaltungen ist die Testentwicklung jedoch einfach und schematisch durchführbar:

Elementare Gatterstrukturen: Dies gilt insbesondere für elementare Gatter wie z.B. das UND-Gatter mit n Eingängen in Bild 3.41a. Der Testvektorsatz für diese Schaltung ist offensichtlich. Ein Fehler f=k/1, k=1..n kann mit Hilfe eines Vektors detektiert werden, der alle Eingangsnetze auf "1" und lediglich Netz k auf "0" setzt. Jeder Fehler f=k/0 kann für alle Werte von k durch einen Testvektor detektiert werden, der alle Eingangsnetze auf "1" setzt (siehe Bild 3.41b).

a)



b)

Vektor Nummer	Vektor $X_1..X_n$	Fehler
1	(0,1,1,....,1)	1/1
2	(1,0,1,....,1)	2/1
⋮	⋮	⋮
n	(1,1,1,....,0)	n/1
n+1	(1,1,1,....,1)	k/0 ; k=1..n

Bild 3.41: UND-Gatter mit n Eingängen
a) als elementare kombinatorische Schaltung
b) Testvektorsatz für die Schaltung

Fan-Out-freie Schaltungen: Vollständig Fan-Out-freie Schaltungen sind eine einfache Erweiterung elementarer Gatter. In einer solchen Schaltung existiert für ein Primäreingangssignal genau ein Pfad zum Primärausgang (siehe Bild 3.42). Ein Test für einen Stuck-at-Fehler an einem Primäreingang kann leicht angegeben werden, da die Ermittlung der Sensibilisierungsbedingungen durch Rückwärtsverfolgung vom Primärausgang zu den Primäreingängen einfach möglich ist. Da für jeden Primäreingang genau ein Pfad zum Primärausgang führt, ist jeder Test für einen Stuck-at-Fehler am Primäreingang auch gleichzeitig ein Test für einen Stuck-at- Fehler auf jedem Netz des Signalwegs zum Primärausgang.

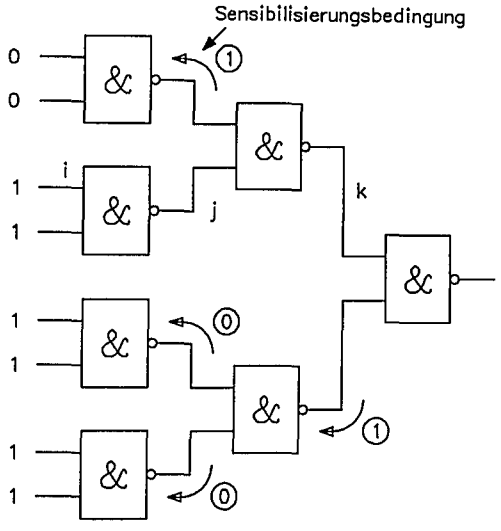


Bild 3.42: vollständig Fan-Out-freie kombinatorische Schaltung mit Sensibilisierung und Test für den Fehler f=i/0

Zweistufige Logik: Die Vorgehensweise bei der Testentwicklung für zweistufige Logik sei anhand der dafür typischen UND-ODER-Struktur nach Bild 3.43 erläutert. Ihre logische Funktion läßt sich durch

$$Z[X]=\overline{X}_1\overline{X}_3 + \overline{X}_3X_4 + X_1X_2X_4 + X_2X_3\overline{X}_4$$

beschreiben. Z nimmt den Wert "1" an für die folgende Menge von Vektoren (1-Würfel);

$$V(Z)=\{0x0x;xx01;11x1;x110\} \quad \text{(Bitreihenfolge } X_1..X_4)$$

Die Schaltung ist also auch durch diese Angabe beschrieben. Der Fehler f=11/0 maskiert den Primimplikanten $\overline{X}_1\overline{X}_3$ (Gatter UND1) bzw. dessen 1-Würfel 0x0x. Die Schaltungsfunktion ist im Fehlerfalle also durch

$$V(Z_f)=\{xx01;11x1;x110\}$$

beschrieben.

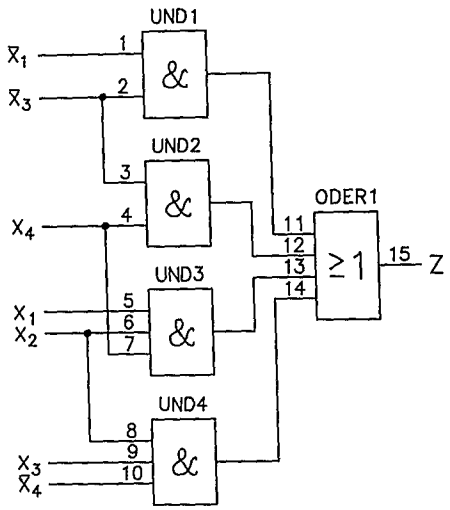


Bild 3.43: Zur Testentwicklung bei zweistufiger Logik

Tabelle 3.1: Ermittlung der Stuck-At-0 Tests der Gatterausgänge der Schaltung von Bild 3.43

	1-Würfel	0-Würfel	Ausgangs-Stuck_at_0-Test
UND1	0X0X	1XXX, XX1X	0000, 0100
UND2	XX01	XX1X, XXX0	1001
UND3	11X1	0XXX, X0XX, XXX0	1111
UND4	X110	X0XX, XX0X, XXX1	0110, 1110

Da ein Testvektor X_t für den Fehler f unterschiedliche Ergebnisse für die fehlerfreie kombinatorische Funktion Z[X] bzw. fehlerhafte Kombinatorik Z_f [X] ergeben muß, kann der Testvektor entweder in V(Z) oder in $V(Z_f)$, nicht aber gleichzeitig in beiden enthalten sein. Ein Testvektor für f=11/0 muß deshalb in der Menge

$$V_t = \{V(Z) - V(Z_f)\} = \{0x0x\}$$

enthalten sein. Hiervon sind andererseits nur diejenigen Vektoren Tests für f=11/0, die echte Minterme (0-Würfel) von Z_f sind, d.h. bei denen die Gatter UND2..UND4 "0" liefern. Als einfaches Ergebnis folgt also:

$$T(11/0) = \{0000,0100\}$$

Dieses Beispiel läßt sich zu einer Testentwurfsregel für Stuck-at-0-Ausgangsfehler von UND-Gattern in zweistufigen UND-ODER-Schaltungen verallgemeinern: Tests sind alle die 0-Würfel aller anderen Gatter, die im 1-Würfel des betrachteten Gatters und in keinem anderen 1-Würfel von Z[X] enthalten sind. Dementsprechend sind T(11/0) = (0000,0100), T(12/0)=(1001), T(13/0)=(1111), T(14/0)= (0110,1110) die Tests für die Schaltung nach Bild 3.43 (vgl. Tabelle 3.1). Beispielsweise ist der Test 1110 ein 1-Würfel des Primimplikanten x110 (1-Würfel des Gatters UND4), nicht aber der anderen. Andererseits ist 1110 in den 0-Würfeln der Gatter UND1, UND2 und UND3 enthalten. Deshalb ist 1110 ein Test für den Fehler f=14/0 der Schaltung nach Bild 3.43.

Bei einem Fehler f=1/1 wird der Primimplikant 0x0x (Gatter UND1) zu xx0x modifiziert. Die fehlerhafte kombinatorische Funktion ist damit durch die Menge der 1-Würfel

$$\mathbf{V}(Z_f)=\{xx0x;xx01;11x1;x110\}$$

beschreibbar. Mögliche Tests müssen deshalb wieder in $\mathbf{V}=\{\mathbf{V}(Z_f) - \mathbf{V}(Z)\}$, nicht aber in V(Z) enthalten sein. Da

$$\mathbf{V}=\{\{xx0x,xx01,11x1,x110\}-\{0x0x,xx01,11x1,x110\}\}=\{1x0x\}$$

ist, ist der Test T(1/1)=(1000) ein möglicher Test für den Fehler f=1/1.

In Verallgemeinerung gilt: für jedes UND-Gatter mit k Eingängen, das den Primimplikanten P realisiert, erzeugt ein Stuck-at-1-Fehler f=i/1 an einem Eingang i einen neuen (fehlerhaften) Primimplikanten P_f (1-Würfel), bei dem an Position i statt "0" bzw. "1" "x" steht. Jedes Element der zugehörigen 0-Würfel, das nicht gleichzeitig in V(Z) enthalten ist, ist ein Testvektor für den Fehler f=i/1.

Da Stuck-at-0-Fehler an einem Eingang eines UND-Gatters äquivalent zu einem Stuck-at-0-Fehler am Gatterausgang sind, können mit diesen beiden Regeln minimale Testvektorsätze für zweistufige UND-ODER-Strukturen ermittelt werden.

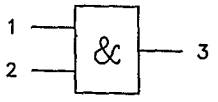
Allgemeine kombinatorische Schaltungen: Für allgemeinere kombinatorische Schaltungsstrukturen existieren eine Reihe von Algorithmen zur Generierung von Testvektoren. Hier sei nur der D-Algorithmus kurz an einem Beispiel vorgestellt.

Der D-Algorithmus wendet die Grundregel eines jeden Tests formal an: 1. Erzeugung einer Pegeländerung auf dem fehlerhaften Netzsegment und 2. Ausbreitung der Änderung bis zu einem Primärausgang.

Beim D-Algorithmus wird das Symbol D und das Komplement $\overline{D}$ eingeführt. D bezeichnet ein Netzsegment, das im fehlerfreien Fall eine "1" führt. Entsprechend bezeichnet $\overline{D}$ ein Netzsegment, das im fehlerfreien Fall eine "0" führt. Der Algorithmus operiert auf dem Wertevorrat $\{0;1;x;D;\overline{D}\}$. Jedes Netzsegment kann einen dieser Werte annehmen. Einem fehlerhaften Netzsegment wird je nach Fehlerart D oder $\overline{D}$ zugeordnet. Anschließend wird die Schaltungsstruktur und der Algorithmus benützt, um andere Netze so zu setzen, daß D oder $\overline{D}$ zu einem Primärausgang sensibilisiert wird.

Zu diesem Zweck muß ein Gatter durch D-Ausbreitungs-Würfel modelliert werden. Ein D-Ausbreitungs-Würfel beschreibt die Fortpflanzung von D bzw. $\overline{D}$ durch das Gatter. Dies sei am UND-Gatter in Bild 3.44 erläutert. Ein D auf Eingang 1 und eine "1" auf Eingang 2 propagiert D zum Ausgang 3, da sich eine fehlerhafte "0" auf Eingang 1 in diesem Fall in einer fehlerhaften "0" am Ausgang bemerkbar machen würde. (D,1,D) ist also ein D-Ausbreitungs-Würfel für dieses Gatter.

Als primitiver D-Würfel eines Fehlers i/0 (oder i/1) auf einem Ausgang i wird ein D-Würfel bezeichnet, dessen Ausgang i mit D (oder $\overline{D}$) bezeichnet ist und an dessen Eingängen solche Pegel vermerkt werden, daß das Gatter "1" (oder "0") am Ausgang i produzieren würde. Der Fehler f=3/0 des Gatters in Bild 3.44 ist deshalb durch den primitiven D-Würfel (1,1,D) beschrieben.



1	2	3
D	1	D
1	D	D
D	D	D
$\overline{D}$	1	$\overline{D}$
1	$\overline{D}$	$\overline{D}$
$\overline{D}$	$\overline{D}$	$\overline{D}$

Bild 3.44: D-Ausbreitungs-Würfel eines 2fach-UND-Gatters

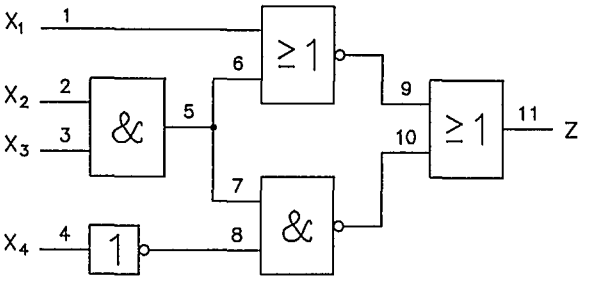


Bild 3.45: Einfache kombinatorische Schaltung zur Illustration des D–Algorithmus

Die Gatterbeschreibung mit D–Würfeln kann unter Zuhilfenahme der Schaltungsstruktur zu einer Tabellenform verbunden werden. Die Netzsegmente der Schaltung von Bild 3.45 sind durchnumeriert. Die Netzsegmentnummern sind die Spaltennummern in Tabelle 3.2. In jede Zeile der Tabelle ist ein D–Ausbreitungs–Würfel eines Gatters eingetragen. Zur Gewinnung eines Tests für einen Fehler wird mit dessen primitiven D–Würfel gestartet und die Propagation von D mit Hilfe dieser Tabelle untersucht. Zur Erläuterung sei der Stuck–at–0–Fehler f=5/0 des UND–Gatters herangezogen, der durch den neuen primitiven

D–Würfel 25: (x 1 1 x D x x x x x x)

beschrieben wird. Tabelle 3.2 zeigt, daß D zu den Netzsegmenten 6 und 7 propagiert. Da das Ziel sein muß, einen Weg zum Primärausgang (Netzsegment 11) zu finden, muß dieser neue D–Würfel (x 1 1 x D D D x x x x) weiter propagieren. Eine weitere Ausbreitung wird durch den D–Würfel 9 (0 x x x x D x x $\overline{D}$ x x) möglich. Die bisherigen D–Würfel können nun zu einem neuen D–Würfel zusammengefaßt werden

D–Würfel 26: (0 1 1 x D D D x $\overline{D}$ x x),

der die Propagation des Fehlers bis zu den Netzsegmenten 7 und 9 beschreibt. Diese beiden Netzsegmente bilden die bisherige D–Ausbreitungsfront.

Tabelle 3.2: Zur Analyse der Fehlerausbreitung in der Schaltung nach Bild 3.45

	Netzsegment #											
Würfel #	1	2	3	4	5	6	7	8	9	10	11	Gatter
1		1	D		D							UND
2		D	1		D							
3		D	D		D							
4		1	$\overline{D}$		$\overline{D}$							
5		$\overline{D}$	1		$\overline{D}$							
6		$\overline{D}$	$\overline{D}$		$\overline{D}$							
7				D				$\overline{D}$				INV
8				$\overline{D}$				D				
9	0					D			$\overline{D}$			NOR
10	D					0			$\overline{D}$			
11	0					$\overline{D}$			D			
12	$\overline{D}$					0			D			
13							1	D		$\overline{D}$		NAND
14							D	1		$\overline{D}$		
15							D	D		$\overline{D}$		
16							1	$\overline{D}$		D		
17							$\overline{D}$	1		D		
18							$\overline{D}$	$\overline{D}$		D		
19									0	D	D	ODER
20									D	0	D	
21									0	$\overline{D}$	$\overline{D}$	
22									$\overline{D}$	0	$\overline{D}$	
23					D	D	D					Netz
24					$\overline{D}$	$\overline{D}$	$\overline{D}$					

Zur Erreichung des Primärausgangs kann der D-Würfel 22 (x x x x x x x x $\overline{D}$ 0 $\overline{D}$) herangezogen werden, so daß die Fehlerpropagation bis zum Primärausgang jetzt insgesamt durch den

D-Würfel 27: (0 1 1 x D D D x $\overline{D}$ 0 $\overline{D}$)

beschrieben wird.

Da der Primäreingang 4 damit aber noch nicht festgelegt ist, muß die Testkonsistenz untersucht werden. Für diese Analyse wird eine Tabelle ähnlich wie Tabelle 3.2 aufgebaut, in der die Wahrheitstabellen der Gatterelemente eingetragen sind (siehe Tabelle 3.3). Der D-Würfel 27 fordert für Netzsegment 10 eine "0". Tabelle 3.3 verlangt hierfür für die Netzsegmente 7 und 8 eine "1". Andererseits enthält der D-Würfel 27 für Netzsegment 7 ein D. Dies zeigt an, daß dieser Würfel keinen konsistenten Test darstellt, der Fehler f=5/0 also nicht über den untersuchten Weg 5/6/9/11 testbar ist. Statt dessen kann der alternative Weg 5/7/10/11 auf dieselbe Weise untersucht werden.

Dies führt schließlich auf den

D-Würfel 28: (x 1 1 x D D D 1 0 $\overline{D}$ $\overline{D}$).

Hier sind die Primäreingänge 1 und 4 noch nicht festgelegt. Für das Netzsegment 8 wird hier "1" gefordert, das gemäß Tabelle 3.3 mit "0" auf Netzsegment 4 gleichbedeutend ist. Hieraus folgt zunächst (x 1 1 0 D D D 1 0 $\overline{D}$ $\overline{D}$). Für das Netzsegment 9 wird "0" gefordert. Würfel 7 von Tabelle 3.3 zeigt, daß dies erreichbar ist, wenn das Netzsegment 1 "1" wird. Damit ist der D-Würfel 28 in der Form (1 1 1 0 D D D 1 0 $\overline{D}$ $\overline{D}$) konsistent und (1110) ist ein Testvektor für den Fehler f=5/0:

T(5/0)=(1110).

Tabelle 3.3: Wahrheitstabelle der Gatter in der Schaltung nach Bild 3.45

	Netzsegment #											
Würfel #	1	2	3	4	5	6	7	8	9	10	11	Gatter
1		0	X		0							UND
2		X	0		0							
3		1	1		1							
4				0				1				INV
5				1				0				
6	0					0			1			NOR
7	1					X			0			
8	X					1			0			
9							0	X		1		NAND
10							X	0		1		
11							1	1		0		
12									0	0	0	ODER
13									1	X	1	
14									X	1	1	
15						0	0	0				Netz
16						1	1	1				

Sequentielle Schaltungen: Bei kombinatorischen Schaltungen testet ein Vektor an den Primäreingängen die Schaltung auf das Vorhandensein eines (evtl. mehrerer) Fehlers. Bei sequentiellen Schaltungen wird i.allg. eine Folge von Testvektoren benötigt, um einen Fehler festzustellen. Mit einer ersten Folge von Testvektoren werden die Speicherelemente der sequentiellen Schaltung (z.B. Flipflops) in einen bekannten Zustand versetzt. Weitere Testvektoren sind anschließend erforderlich, um mögliche Fehler zu den Primärausgängen zu bringen und damit feststellbar zu machen. Dies ist bereits eine beträchtliche Komplikation.

Schwerwiegender ist jedoch, daß Fehler in einer Schaltung verhindern können, einzelne Speicherelemente überhaupt in einen bekannten Zustand zu versetzen und gar unmöglich machen, überhaupt Testbedingungen für einen Fehler einzustellen. Fehler können außerdem zu unerwarteten Zustandsübergängen (Oszillationen) in diesen sequentiellen Schaltungen führen. Eine fehlerfreie Schaltung kann frei von Signalwettläufen (races) oder gefährlichen Transienten (hazards) entwickelt sein. Bei Fehlern in der Schaltung können solche jedoch erneut auftreten. Ein Fehler kann die Zahl der Zustände in einer Schaltung ändern. Kombinatorische Schaltungen können durch Fehler sequentiell werden. Fehler können aus synchronen Schaltungen asynchrone machen und umgekehrt.

Funktioneller Test: Es ist offensichtlich, daß mit wachsender System- oder Schaltungsgröße die Komplexität der Testvektorentwicklung drastisch zunimmt. Deshalb wird immer wieder der pragmatische Ansatz unternommen, Systeme, Schaltungen oder IC funktionell zu testen, d.h. die Eingänge mit Signalen zu beaufschlagen und so zu verifizieren, daß die gewünschte Funktion durch die Schaltung tatsächlich erfüllt wird. Die Beurteilung der Testgüte ist schwer möglich. Auch wenn die Sollfunktion mit solchen Tests überprüft wird, besteht trotzdem die Möglichkeit, daß die fehlerbehaftete Schaltung zusätzliche unerwünschte Verhaltensweisen aufweist, die mit dem Funktionstest nicht festgestellt werden. Wenn beispielsweise ein Multiplexer aufgrund eines Fehlers zusätzlich zur korrekten Eingangsleitung eine weitere Leitung gleichzeitig selektiert, kann dies allenfalls nur dann festgestellt werden, wenn die korrekte Leitung eine "0" und die fälschlicherweise selektierte Leitung eine "1" trägt.

Ein funktioneller Test muß also nicht nur nachweisen, daß die gewünschte Funktion erfüllt wird, sondern auch daß keine zusätzlichen unerwünschten Funktionen ausgeführt werden. Der offensichtliche Ansatz zur Lösung dieses Problems ist deshalb der erschöpfende Test (exhaustive test). Dabei werden alle kombinatorischen Möglichkeiten der Eingänge durchgespielt und die Ausgangsfunktionen auf Fehlerfreiheit geprüft. Bei 16 Eingängen sind dies 65536 Möglichkeiten, eine Größenordnung, die ohne Schwierigkeiten begehbar ist. Falls aber die Schaltung noch 30 Flipflops enthält, sind insgesamt $2^{(16+30)}$ Kombinationen möglich. Selbst wenn ein Einzeltest nur 10 ns benötigen würde, würde der Test etwa acht Tage beanspruchen, von der zu bewältigenden Datenmenge von ca. 15 Terabyte ganz abgesehen.

Um Tests handhabbarer Länge für komplexe Schaltungen entwickeln zu können, wird Kenntnis über die Strukturen der Schaltung und das erwartete Fehlverhalten beim Vorliegen von physikalischen Fehlern benötigt. Beispielsweise würden für den Test eines 32-Bit-Addierers ohne weitere Strukturkenntnis 2^{65} Testvektoren nötig werden (65 = 2*32 Dateneingänge + 1 Übertragseingang). Ist jedoch bekannt, daß der 32-Bit-Addierer aus 32 1-Bit-Ripple-Carry-Volladdierern aufgebaut ist, so kann jeder Volladdierer mit 8 Tests geprüft werden. Der Gesamttest benötigt dann nur noch etwa 32*8=256 Tests.

3.6 Entwurf testbarer Schaltungen

3.6.1 Einführung

Die in Abschnitt 3.5 vorgestellten Prinzipien und Probleme des Tests digitaler integrierter Schaltungen machen die Komplexität dieses Entwicklungsschritts deutlich. Große Forschungsanstrengungen werden unternommen, um bessere Testverfahren zu finden. Es ist aber unwahrscheinlich, daß Techniken entwickelt werden, mit denen einfache ökonomische Tests für beliebige Schaltungen möglich sind. Vielmehr ist es wahrscheinlicher, daß ökonomische Tests nur für Schaltungsentwürfe durchführbar sind, die explizite Testhilfen enthalten. Mit den Begriffen "Testfreundlicher Entwurf", "Testfreundlichkeit" oder "Design for Testability" werden Entwurfstechniken umschrieben, die den Test eines entstehenden Produkts später vereinfachen oder überhaupt erst ermöglichen.

Testability ist dabei ein unpräziser Fachbegriff, da in die Testaufwendungen bzw. -kosten zahlreiche Faktoren eingehen. Die Testvektorentwicklung erfordert üblicherweise sehr große Rechenzeiten und verursacht neben deren Kosten auch Kosten für die Entwicklung der Testprogramme und Kosten für die verlängerten Entwicklungszeiten für eine Schaltung bzw. das Produkt.

Auch die Testdurchführung ist mit großen Kosten behaftet. Testautomaten stellen Millioneninvestitionen dar. Je mehr Zeit der Test eines integrierten Schaltkreises auf einem Tester beansprucht, wirkt sich das auf den Preis des Bausteins aus. So ist ein Kostenoptimum zu finden, das möglicherweise eingesparte Testkosten gegen die Kosten späterer Folgen nicht erkannter Fehler abwägt. Wird ein Chipfehler erst nach Montage einer gedruckten Leiterplatte erkannt, muß möglicherweise die gesamte Leiterplatte als Ausschuß abgeschrieben werden.

Die Testbarkeit eines Produkts ist offensichlich dann besser geworden, wenn im Vergleich zu vorher die Kosten für die Testentwicklung und die Testdurchführung gesenkt werden oder/und der Anteil der erkennbaren Fehler (Fehlerabdeckung) und Fehlerdiagnosemöglichkeiten wachsen.

3.6.2 Testbarkeitsmaße

Wie in Abschnitt 3.5 ausgeführt wurde, erfordert ein Test die Erzeugung einer Pegeländerung an der zu testenden Stelle innerhalb der Schaltung mit Hilfe der Primäreingänge und die Sensibilisierung eines Primärausgangs für diese Änderung. Ein Ansatz für die Bewertung der Testbarkeit einer Schaltung wäre, mit Hilfe eines ATPG (*A*utomatic *T*est *P*attern *G*eneration)-Programms Testvektoren generieren zu lassen und die Fehlerabdeckung als Testbarkeitskriterium zu nehmen. Die Fehlerabdeckung wäre der Prozentsatz aller Fehler (unter Zugrundelegung eines Fehlermodells), der durch den Testvektorsatz erkannt wird.

Solche Rechenläufe sind sehr langandauernd und äußerst kostenträchtig. Der Hauptnachteil liegt jedoch darin, daß sich daraus meistens nur wenige Hinweise ergeben, wie die Testbarkeit verbessert werden könnte. Deshalb wurden einige Programme entwickelt, die Bewertungsmaße für die Testbarkeit ermitteln. Alle diese Programme basieren entsprechend Abschnitt 3.5 auf dem Konzept der Kontrollierbarkeit und Beobachtbarkeit für jedes Schaltungselement.

Kontrollierbarkeit ist dabei ein Aufwandsmaß für die Erzeugung eines beliebigen gültigen Signalpegels an den Eingängen einer Schaltungskomponente innerhalb der Schaltung mit Hilfe der Primäreingänge. Beobachtbarkeit ist ein Aufwandsmaß für das Sichtbarmachen einer Signaländerung des Ausgangs einer Schaltungskomponente innerhalb der Schaltung an den Primärausgängen.

Als Beispiel sei kurz auf SCOAP eingegangen (SCOAP = Sandia Controllability Observability Analysis Program). In SCOAP werden Schaltungsknoten als kombinatorisch oder sequentiell betrachtet. Ein kombinatorischer Knoten ist entweder ein Primäreingang oder der Ausgang eines kombinatorischen Logikelements (logische Zelle). Ein sequentieller Knoten ist der Ausgang eines sequentiellen Logikelements (z.B. Flipflops). Jedem Knoten ist die Eigenschaft Kontrollierbarkeit/Beobachtbarkeit als ein Vektor aus den sechs Elementen zugeordnet:

CC0(n): kombinatorische 0-Kontrollierbarkeit, das ist die minimale Zahl von Pegelfestlegungen anderer Knoten, um den Knoten n auf "0" zu setzen.

CC1(n): kombinatorische 1-Kontrollierbarkeit, das ist die minimale Zahl von Pegelfestlegungen anderer Knoten, um den Knoten n auf "1" zu setzen.

SC0(n): sequentielle 0-Kontrollierbarkeit, das ist die minimale Zahl sequentieller Knoten, die auf spezifische Werte gesetzt werden müssen, um den Knoten n auf "0" zu setzen.

SC1(n): sequentielle 1-Kontrollierbarkeit, das ist die minimale Zahl sequentieller Knoten, die auf spezifische Werte gesetzt werden müssen, um den Knoten n auf "1" zu setzen.

CO(n): kombinatorische Beobachtbarkeit, das ist die Zahl kombinatorischer Logikelemente zwischen dem Knoten n und einem Primärausgang und die minimale Zahl kombinatorischer Knoten, die gesetzt werden müssen, um den Beobachtungsweg zum Primärausgang zu öffnen.

SO(n): sequentielle Beobachtbarkeit, das ist die Zahl sequentieller Logikelemente zwischen dem Knoten n und einem Primärausgang und die minimale Zahl sequentieller Knoten, die gesetzt werden müssen, um den Pegelwert des Knotens n zum Primärausgang fortzuschalten.

Daneben ist jedem Logikelement ein Satz von Regeln zugeordnet, mit denen die Kontrollierbarkeiten eines Ausgangs aus den Kontrollierbarkeiten der Eingänge ermittelt werden. Beispielsweise gilt für ein zweifach UND-Gatter mit den

Eingängen X_1,X_2 und dem Ausgang Y

$$CC0(Y) := \min[CC0(X_1),CC0(X_2)] + 1 ,$$

da es offensichtlich genügt, einen Eingang auf "0" zu kontrollieren, um den Ausgang Y auf "0" zu setzen. Die Addition von 1 berücksichtigt, daß ein weiteres kombinatorisches Element durchlaufen wurde. Die restlichen Regeln für das UND–Gatter sind:

$$CC1(Y) := CC1(X_1) + CC1(X_2) + 1$$
$$SC0(Y) := \min[SC0(X_1),SC0(X_2)]$$
$$SC1(Y) := SC1(X_1) + SC1(X_1) .$$

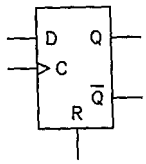
Die Beobachtbarkeiten der Eingänge lassen sich mit den folgenden Regeln ermitteln:

$$CO(X_1) := CO(Y) + CC1(X_2) + 1$$
$$CO(X_2) := CO(Y) + CC1(X_1) + 1$$
$$SO(X_1) := SO(Y) + SC1(X_2)$$
$$SO(X_2) := SO(Y) + SC1(X_1)$$

Etwas komplexer, aber trotzdem leicht einsichtig, sind die entsprechenden Regeln für das Beispiel eines D–Flipflops (siehe Bild 3.46). Um z.B. den Ausgang Q auf "0" zu kontrollieren, kann der Reset–Eingang R auf "1" und der Takteingang auf "0" kontrolliert werden. Alternativ kann der D–Eingang auf "0", der Reset–Eingang auf "0" und der Takteingang erst auf "0" und dann auf "1" kontrolliert werden.

Zur Ermittlung der Kontrollierbarkeit der Schaltungsknoten wird bei den Primäreingängen mit den Startwerten CC0(x)=CC1(x)=1, SC0(x)=SC1(x)=0 begonnen. Zur Ermittlung der Beobachtbarkeitwerte wird bei den Primärausgängen mit den Startwerten CO(x)=SO(x)=0 begonnen. Bild 3.47 zeigt eine einfache Schaltung und die zugehörigen kombinatorischen Testbarkeitsmaße. Die 0–Kontrollierbarkeit von Q1 ist mit dem Wert 64 am schlechtesten, während die Beobachtbarkeit des Knotens NO1 mit dem Wert 36 am schlechtesten ist. Mit diesen Testbarkeitsmaßen sind gezielt die Schaltungsteile identifizierbar, die einem Test über die Primärein– bzw. –ausgänge schwer zugänglich sind.

Der Schaltungsentwickler kann durch geeignete Maßnahmen die Testfreundlichkeit erhöhen. Wird in der Schaltung Bild 3.47 zugelassen, daß CLK ein Ein–/Ausgang, Q3 zusätzlich ein Eingang und NA2 zusätzlich ein Ausgang der Schaltung ist, so reduzieren sich die Testaufwendungen beträchtlich: CC0(Q1)=19 (vorher 64), CC1(NO2)=15 (vorher 57) und CO(NO1)=2 (vorher 36).



$CC0(Q) := \min[CC1(R)+CC0(C);\ CC0(D)+CC0(C)+CC1(C)+CC0(R)]$

$CC1(Q) := CC1(D)+CC0(C)+CC1(C)+CC0(R)$

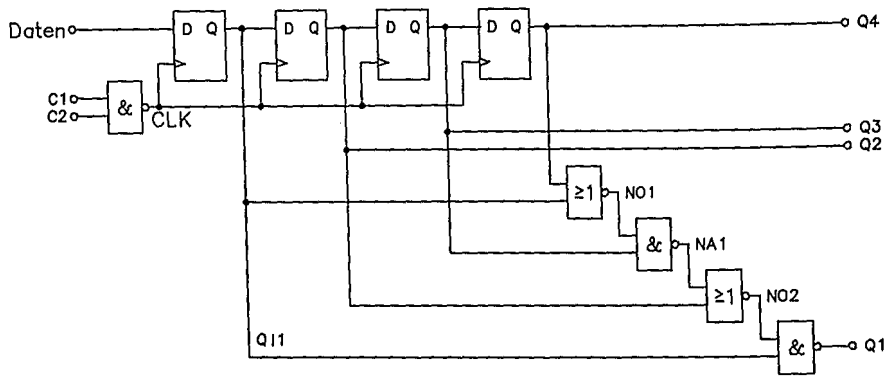
$SC0(Q) := \min[SC1(R)+SC0(C);\ SC0(D)+SC0(C)+SC1(C)+SC0(R)]+1$

$SC1(Q) := SC1(D)+SC0(C)+SC1(C)+SC0(R)+1$

$CO(D) := CO(Q)+CC0(C)+CC1(C)+CC0(R)$

$SO(D) := SO(Q)+SC0(C)+SC1(C)+SC0(R)+1$

Bild 3.46: Einige Kontrollierbarkeits- und Beobachtungsregeln eines D-Flipflops mit Reset-Eingang



	Testbarkeitsmaße		
Netz	CC0	CC1	CO
C1	1	1	19
C2	1	1	19
Daten	1	1	10
NA1	45	8	19
Q1	64	7	0
Clk	3	2	17
NO2	9	57	7
NO1	7	28	36
Q11	6	6	5
Q2	11	11	0
Q3	16	16	0
Q4	21	21	0

Bild 3.47: Einfache Schaltung und deren kombinatorische Testbarkeitsmaße

3.6.3 Ad-hoc-Methoden

Die vorstehenden Ausführungen verdeutlichen, daß ein Schlüssel für die leichtere Testbarkeit in der Verbesserung der Kontrollierbarkeit und Beobachtbarkeit innerer Netze einer Schaltung bzw. eines integrierten Schaltkreises liegt. Eine kleine Auswahl von ad-hoc-Verfahren ist in den folgenden Bildern und den Begleittexten kurz erläutert. Zusätzlich zu diesen Beispielregeln sollte darauf geachtet werden, daß interne Taktgeneratoren in einen Testmode abschaltbar und durch den Takt des Testautomaten ersetzbar sind. Zusätzlich muß es für den Test möglich sein, Rückkopplungszweige in der Schaltung zur Testvereinfachung aufzutrennen.

Nutzung unbenutzter Pins für Testzwecke: Die Beobachtbarkeit interner Netze kann durch Verwendung unbenutzter Pins erhöht werden (Bild 3.48). Die Kontrollierbarkeit eines internen Netzes läßt sich durch den Einsatz unbenutzter Pins verbessern (Bild 3.49).

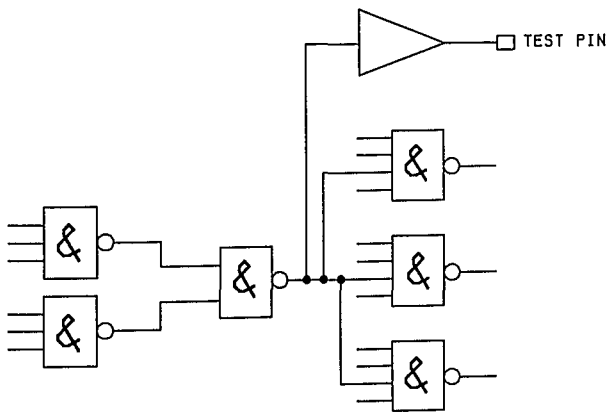


Bild 3.48: Zusätzlich herausgeführtes internes Netz zur Steigerung der Beobachtbarkeit

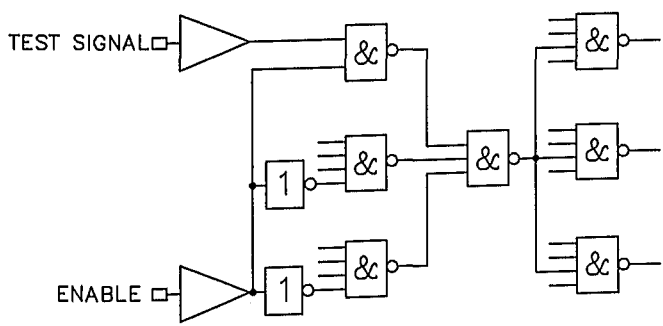


Bild 3.49: Erhöhung der Kontrollierbarkeit interner Netze durch Testpins

Redundante Logik: Bild 3.50 zeigt eine Schaltung mit einem OR-Gatter, das ein Signal auf zwei Wegen erreichen kann. Diese Schaltung ist untestbar, da die beiden Zweige nicht getrennt testbar sind. Zum Test müssen die beiden Zweige auftrennbar sein.

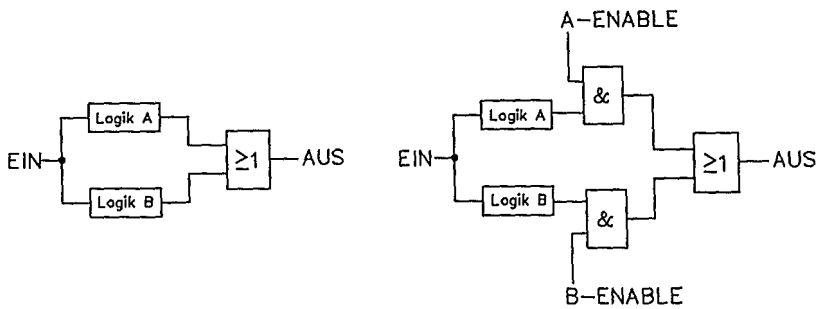


Bild 3.50: Entkopplung redundanter Logik für Testzwecke

Entkopplung verbundener Logikblöcke: Normalerweise verbundene Logikblöcke können für Testzwecke entkoppelt werden (Bild 3.51).

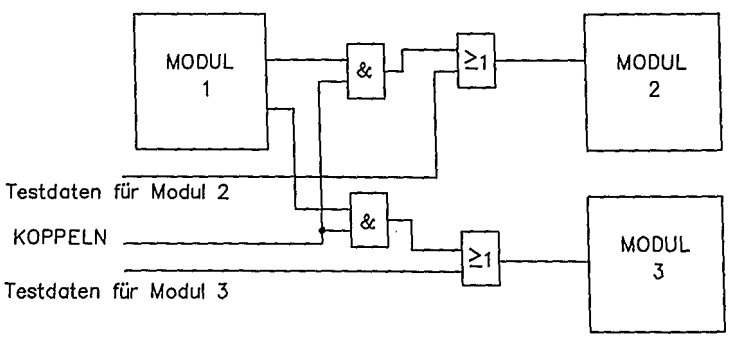


Bild 3.51: Entkopplung von Logikblöcken zur Testvereinfachung

Lange Zählerketten aufbrechen: Lange Zählerketten sollten in kleinere, testbare Module auftrennbar sein (Bild 3.52).

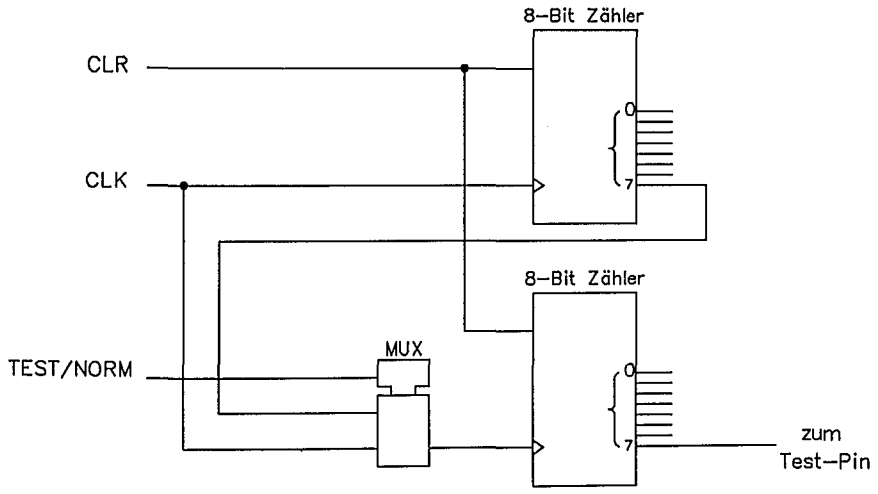


Bild 3.52: Aufbrechen langer Zählerketten

Initialisierung von Zustandsautomaten: Keine Kombination von SELECT und CLOCK kann den Automat von Bild 3.53 in einen bekannten Zustand überführen. Die Schaltung ist untestbar!

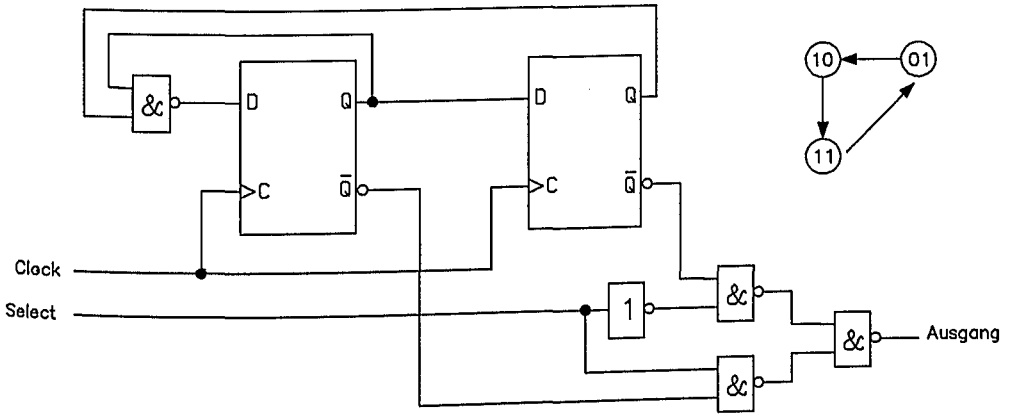


Bild 3.53: Ein nicht-initialisierbarer Zustandsautomat ist nicht testbar

Durch rücksetzbare Flipflops kann ein bekannter Ausgangszustand eingestellt werden (Bild 3.54). Als generelle Regel sollte jeder Schaltungsteil mit Flipflops über externe Pins alleine gesetzt oder rückgesetzt werden können.

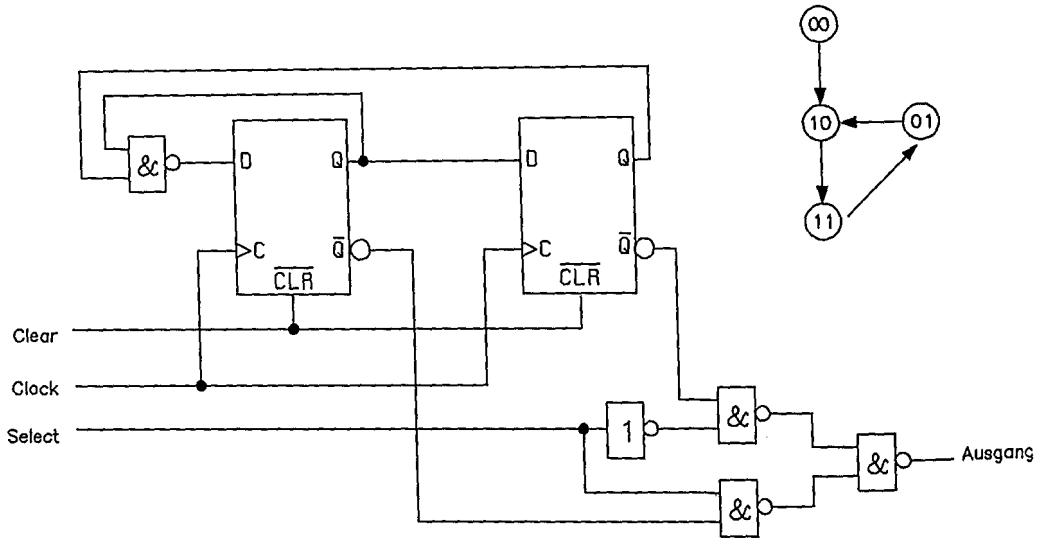


Bild 3.54: Testbarer Zustandsautomat durch Rücksetzeingang

Schieberegister als Testsignalgenerator: Mit drei Pins lassen sich in dem Beispiel von Bild 3.55 bis zu 4096 Testmuster einspeisen. Als Erweiterung können Schieberegister verwendet werden, um die Logikpegel interner Netze über einen Pin beobachtbar zu machen.

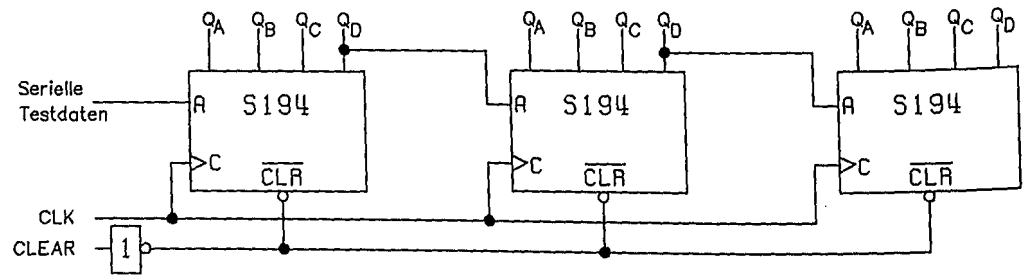


Bild 3.55: Schieberegister als Testsignalgenerator zur Erhöhung der Kontrollierbarkeit

Multiplexer zur Testbarkeitserhöhung: Multiplexer an Ausgangspins ermöglichen die Beobachtung sonst nicht zugänglicher Netze (Bild 3.56). Multiplexer an Eingangspins erlauben die Kontrolle sonst nicht kontrollierbarer Netze (Bild 3.57).

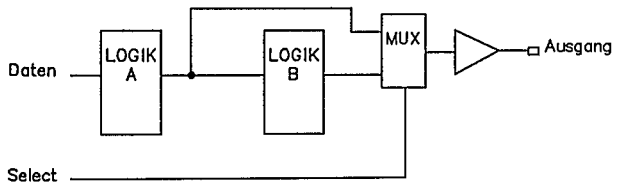


Bild 3.56: Steigerung der Beobachtbarkeit interner Netze durch Ausgangsmultiplexer

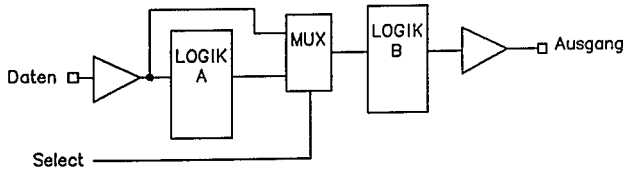


Bild 3.57: Erhöhung der Kontrollierbarkeit interner Netze durch Multiplexer unter Umgehung von Logikblöcken (hier LOGIK A)

Bidirektionale Pins: Die Benützung von bidirektionalen Puffern für sonst unidirektional verwendete Pins erlaubt deren zusätzliche Verwendung zur Kontrolle oder Beobachtung von Netzen. Damit wird die Effektivität der vorhandenen Gehäusepins erhöht (Bild 3.58).

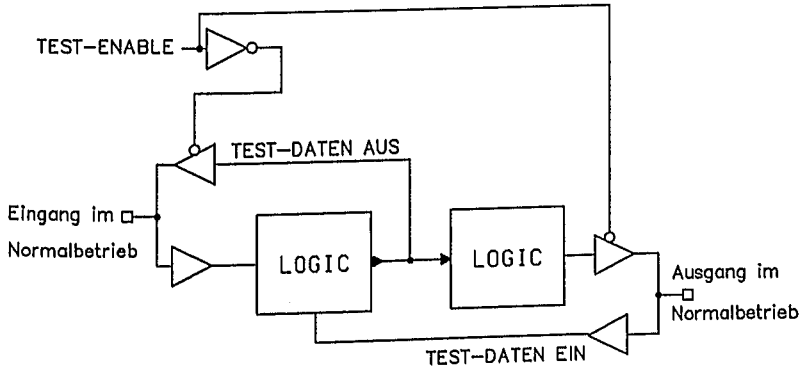


Bild 3.58: Einführung bidirektionaler Pins zur Verbesserung der Testbarkeit

Multiplexer zur Beobachtung versteckter Flipflops: Von den drei Flipflops in Bild 3.59 ist nur eines direkt an einem Gehäusepin beobachtbar. Durch Einsatz eines Multiplexers kann jedes unmittelbar beobachtet werden (Bild 3.60).

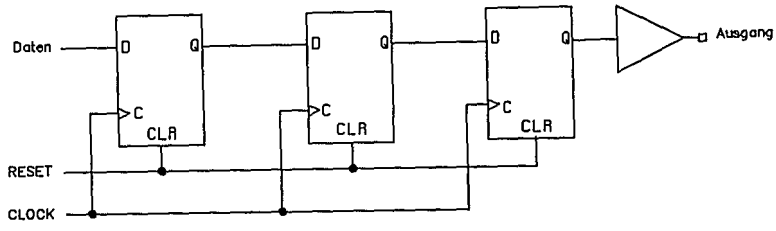


Bild 3.59: Versteckte Flipflops erschweren die Testbarkeit

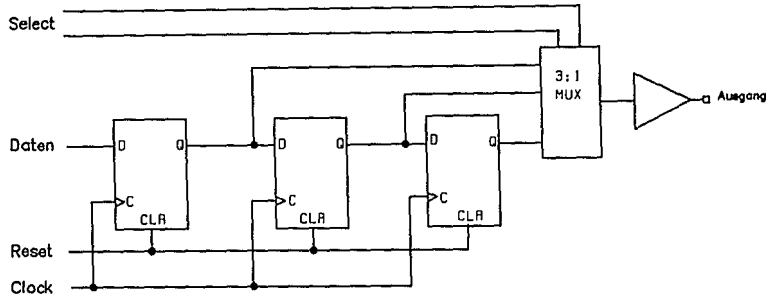


Bild 3.60: Leichtere Beobachtbarkeit versteckter Flipflops durch einen Ausgangsmultiplexer

3.6.4 Scan-Techniken

Eine Hauptschwierigkeit der ad-hoc-Techniken entstammt der Notwendigkeit, zusätzliche Kontrolleingänge und Beobachtungsausgänge vorzusehen. Ein Blick auf die prinzipielle Struktur einer digitalen Schaltung (Bild 3.61) verdeutlicht dieses Problem. Demnach besteht sie aus Bereichen kombinatorischer Logik, die von Registerbereichen (hier Flipflops: FF) getrennt werden. Der Kombinatorikblock A ist zwar gut kontrollierbar, da er unmittelbar über die Primäreingänge erreichbar ist. Seine Ausgänge führen aber zu Registerbereichen.Sie sind nicht ohne weiteres an den Primärausgängen beobachtbar. Zur Beobachtung müßten sie erst durch die Register in die Kombinatorikblöcke B und C eingespeist werden, um (vielleicht) an den Ausgängen beobachtbar zu sein. Offensichtlich ist es besonders schwierig, die Eingänge der Kombinatorik B mit Testvektoren zu beaufschlagen und deren Ausgänge zu beobachten. Unter Anwendung der ad-hoc-Methoden wäre es naheliegend, Netze zwischen den Kombinatorikblöcken über zusätzliche Testlogik kontrollierbar und beobachtbar zu machen. Dies erfordert bei größeren Schaltungen schnell eine große Zahl von zusätzlichen Test-Pins und zusätzlicher Logik.

In den Scan–Path–Techniken wird eine Methode angewandt, die praktisch unabhängig von der Schaltungsgröße mit sehr wenigen, typischerweise ein bis vier zusätzlichen externen Verbindungen auskommt. Dazu wird eine Schaltung so entworfen, daß sie in zwei Betriebsarten arbeiten kann. Im Normalmodus führt sie ihre Sollfunktion aus. Im Testmodus werden die Flipflops der Schaltung als Schieberegister betrieben (siehe Bild 3.62). Über einen Testdateneingang (Scan Daten Ein) können alle Flipflops seriell geladen werden. Jeder kombinatorische Block wird dadurch mit Testeingangsdaten gezielt versorgt. Danach wird die Schaltung für einen Takt in den Normalbetriebszustand gesetzt, um mit der nächsten Taktflanke die Ausgangssignale jedes Kombinatorikblocks in die Flipflops zu laden. Anschließend wird in den Scan–Modus zurückgewechselt, und die Testergebnisse werden seriell zur Auswertung ausgegeben, während gleichzeitig das Scan–Register mit neuen Testdaten seriell neu geladen wird. Im Testmodus wirken also alle Flipflops zusammmen wie ein seriell ladbarer Signalmustergenerator und ein seriell auslesbarer Logikanalysator.

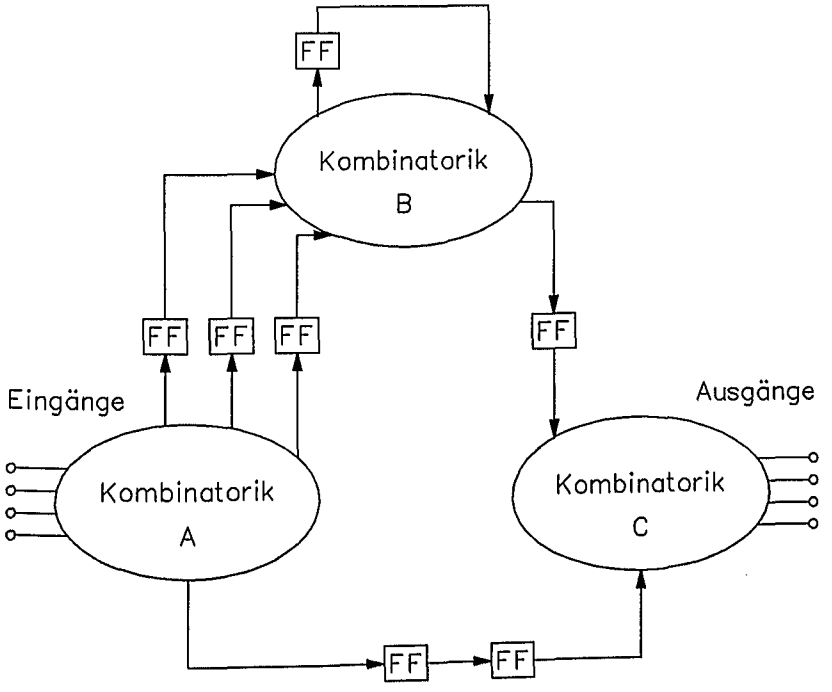


Bild 3.61: Prinzipielle Struktur digitaler Schaltungen und normaler Betriebsmodus

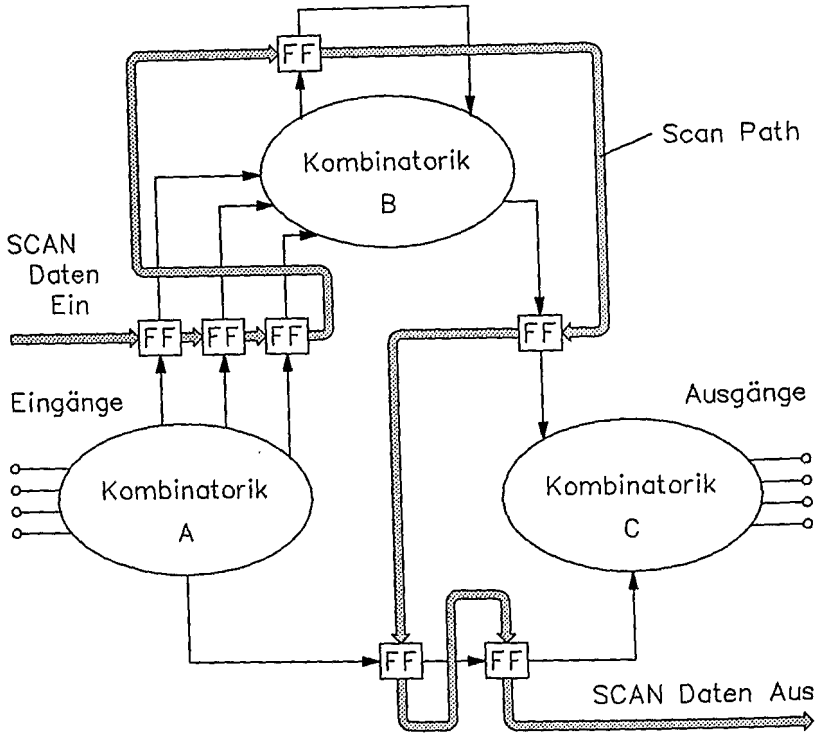


Bild 3.62: Prinzipielle Struktur digitaler Schaltungen im Scan-Path-Test-Modus

3.6.4.1 Scan-Path-Design

Die einfachste Implementierungsmöglichkeit ist in Bild 3.63 dargestellt. Alle Flipflops werden durch einen gemeinsamen Takt weitergeschaltet. Über einen 2:1-Datenselektor, der über das Scan-Modus-Kontrollsignal eingestellt wird, erhält jedes Flipflop entweder das Signal z_i des normalen Betriebsmodus aus der Kombinatorik oder unmittelbar das Ausgangssignal des Vorgänger-Flipflops. In vielen Makrozellbibliotheken ist dieser Multiplexer als Teil sogenannter Scan-Flipflops enthalten. Als Beispiel sei die Makrozelle DF5B aus der 2µm-Gate-Array-Bibliothek von AMS erwähnt. Eine simple Anwendung aus einer Praktikumsübung ist in Bild 3.64 dargestellt. An die PD-Eingänge sind die Normalbetriebsdaten der Schaltung (NextMTCH..NextAUX3) herangeführt. Die Scandata-Eingänge SD sind jeweils mit dem Q-Ausgang des Vorgänger-Flipflops der Scankette verbunden. SD des Schiebekettenkopfes ist mit dem Testdateneingang des Chips verbunden. Der Scan-Enable-Eingang SE schaltet alle Flipflops in Serie, die somit über SCANDATA seriell geladen und über MTCH, ein Ausgangssignal des Chips, seriell ausgelesen werden können.

Die Testdatengenerierung kann für jeden Kombinatorikblock (siehe Bild 3.61) unabhängig unter Anwendung der oben skizzierten Methoden und mit Hilfe von ATPG–Programmen erfolgen. Die Testvektoren für die einzelnen kombinatorischen Blöcke werden aneinandergereiht und seriell geladen bzw. die Testergebnise jedes Testschritts wieder ausgelesen.

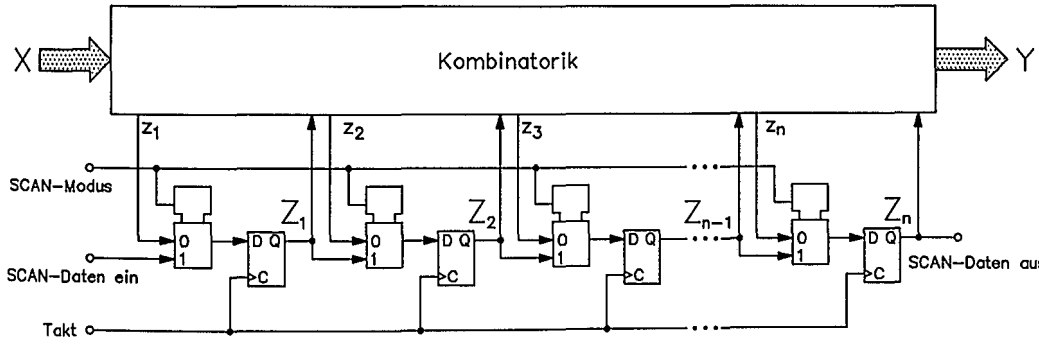


Bild 3.63: Scan–Path–Testability–Design (schematisch)

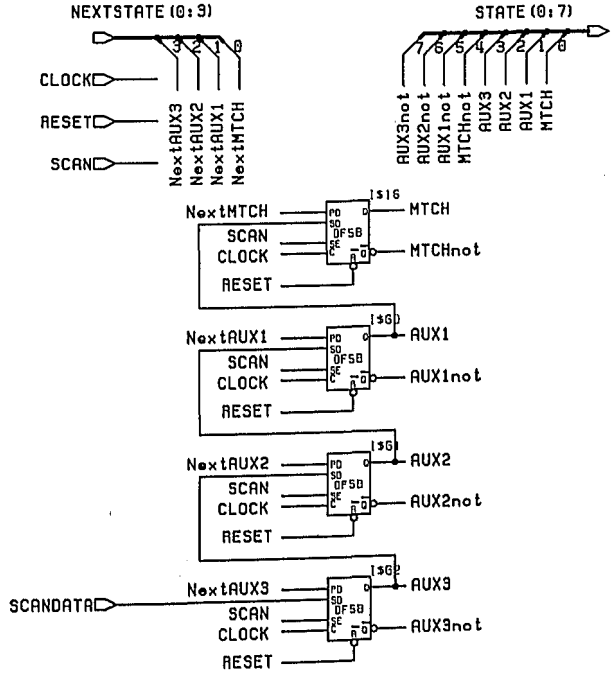


Bild 3.64: Einfaches Anwendungbeispiel für SCAN–Flipflops

3.6.4.2 Level-Sensitive-Scan-Design (LSSD)

LSSD ist eine Scan-Path-Technik unter Einsatz von Latches anstelle von flankengetriggerten Flipflops. Latches können nicht zu Schieberegistern rekonfiguriert werden. Eine generelle LSSD-Struktur ist in Bild 3.65 dargestellt. Während des Normalbetriebs wird das System mit zwei nichtüberlappenden Taktsignalen CK1 und CK2 getaktet. Solange CK1 "1" ist, sind die Latches L_{1-x} transparent und die Latches L_{2-x} halten den vorherigen Logikpegel ihrer Dateneingänge. Die Latches L_{1-x} sind dual-port-Latches. Zum Test werden sie folgendermaßen verwendet:

1. Serielles Einlesen der seriellen Testdaten über den Eingang SDI und abwechselndes Takten des Test-Taktes TCK (genannt LSSD-A-Clock) und des Systemtaktes CK2 (LSSD-B-Clock).
2. Anlegen der parallelen Testdaten an die X-Eingänge.
3. Überprüfen der Ausgangssignale Y der Kombinatorik nach Abwarten der Durchlaufzeit.
4. Anlegen eines Taktimpulses des Systemtaktes CK1 zum Übernehmen der neuen Kombinatorikdaten z in die Latches L_{1-x}.
5. Serielle Ausgabe dieser Daten über SDO durch wechselweises Takten von TCK und CK2.

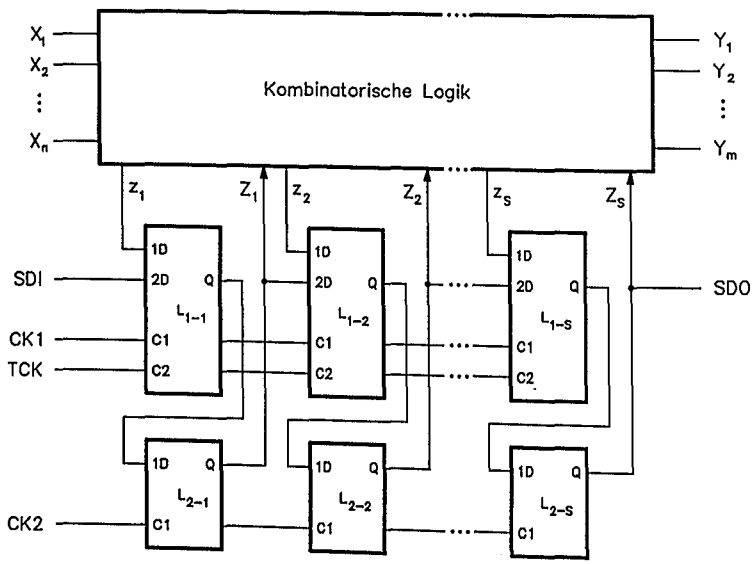


Bild 3.65: Allgemeine Struktur einer Schaltung unter Anwendung des LSSD-Testprinzips

3.6.4.3 Multiplexer-Scan-Strukturen

Die vorstehenden Beispiele benützen Schieberegisterstrukturen für seriell-parallel-seriell-Wandlung von Testdaten. Da die Serienwandlung von parallelen Daten auch über Multiplexer möglich ist, können Teststrukturen wie in Bild 3.66 eingesetzt werden. Über den Multiplexer können die Registerinhalte nacheinander auf den Ausgangspins sichtbar gemacht werden. Dieses Strukturprinzip erhöht also die Beobachtbarkeit, nicht jedoch die Kontrollierbarkeit. Sie wird deshalb seltener angewandt als die Scan-Path-Strukturen.

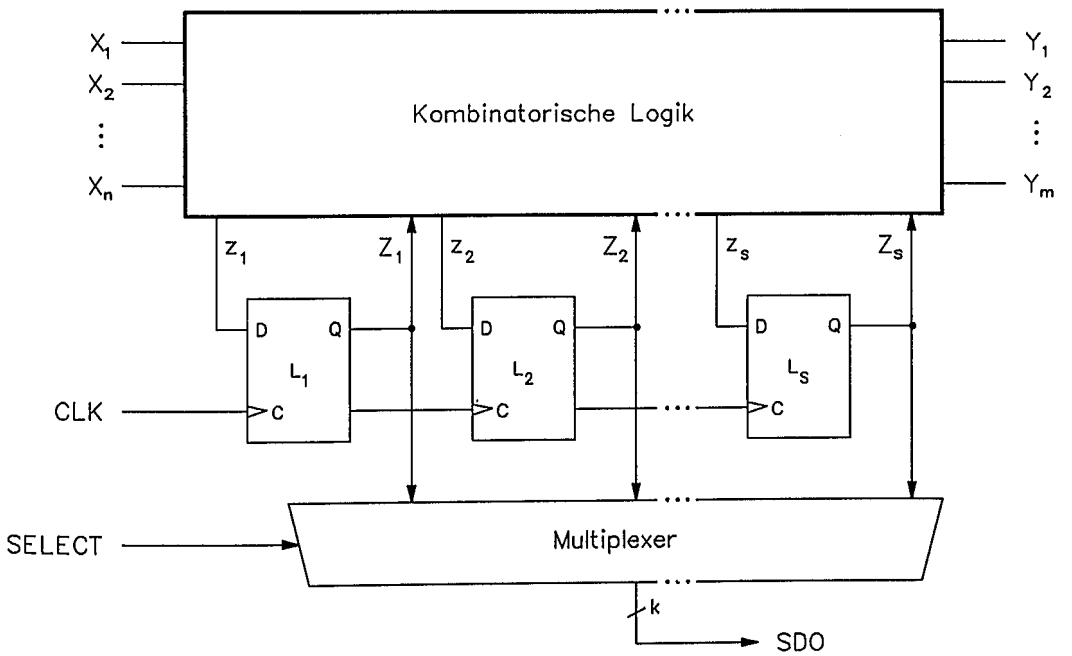


Bild 3.66: Allgemeine Struktur einer Schaltung mit Scan-Testmöglichkeit; die Inhalte der Scan-Flipflops werden über einen Multiplexer sequentiell ausgegeben (SDO)

3.6.4.4 Boundary-Scan-Techniken

Die bisher vorgestellten Techniken erhöhen die Testbarkeit, weil einerseits die Kontrollierbarkeit und Beobachtbarkeit verbessert wird und andererseits die Schwierigkeit eliminiert wird, Testdaten für sequentielle Schaltungen entwickeln zu müssen. Mit den Boundary-Scan-Techniken wird die Testfreundlichkeit dadurch gesteigert, daß die Anschlußerfordernisse an die Testsysteme gemildert werden. Da hierzu jüngst der Standard JTAG/P1149.1 verabschiedet wurde, der auch den Test höherer Systemaggregate unterstützt, sei hier spezifisch auf diese Variante eingegangen. Manche ASIC-Anbieter unterstützen diese Testphilosophie durch Bereitstellen geeigneter Soft-Makrozellen in ihren Bibliotheken. Als Beispiel sei die SCOPE-Cell-Familie von Texas Instruments erwähnt (SCOPE: System Controllability/Observability Partitioning Environment; Markenzeichen von Texas Instruments).

JTAG (Joint Test Action Group) definiert eine Testarchitektur, die auf einem vierdrahtigen Testbus basiert. Er besteht aus: TDI Test Data In, TDO Test Data Out, TCK Test Clock und TMS Test Mode Select. Bild 3.67 zeigt die Anwendung dieser Architektur für Testzwecke.

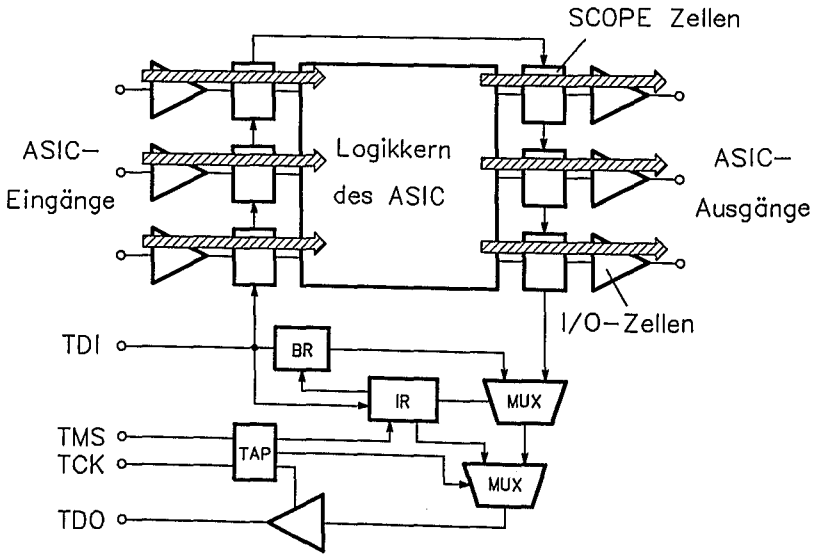


Bild 3.67: Boundary-Scan-Test Prinzip unter Anwendung des JTAG/P1149.1-Testbusses

Um den eigentlichen Logikkern (sequentiell und kombinatorisch) gruppiert sich für jeden Eingang bzw. Ausgang eine Boundary-Scan-Zelle, die hier mit SCOPE-Zelle bezeichnet ist. Die ASIC-Eingangssignale gelangen durch diese zum eigentlichen Kern des Bausteins, bzw. vom Kern zu den ASIC-Ausgängen (Normalbetrieb). Im Boundary-Scan-Test-Modus können beispielsweise Systemdaten an den ASIC-Eingängen oder/und ASIC-interne Testdaten aufgesammelt und seriell über TDO zur Auswertung ausgegeben werden (siehe Bild 3.68). Auf ähnliche Weise können Testvektoren dem Logikkern seriell zugeführt werden (siehe Bild 3.69).

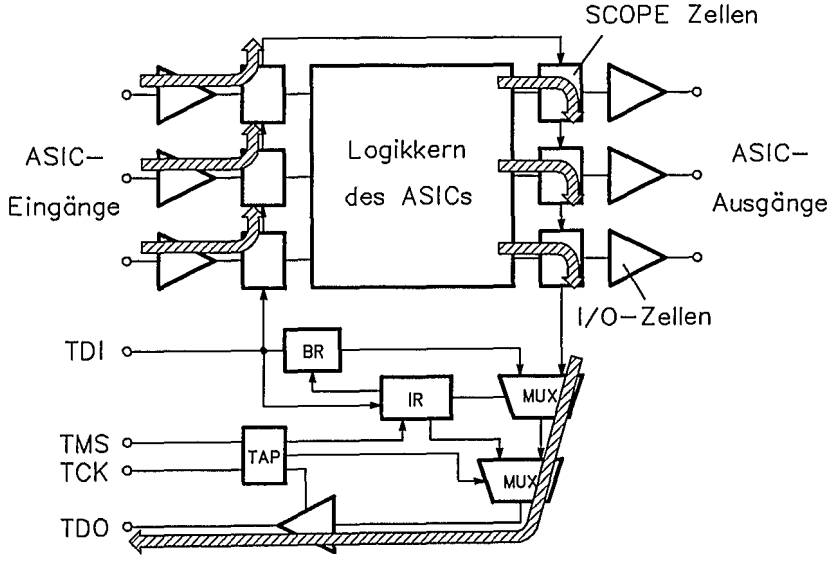


Bild 3.68: Boundary-Scan-Modus: Testdatengewinnung und serielle Ausgabe auf dem Testdatenbus

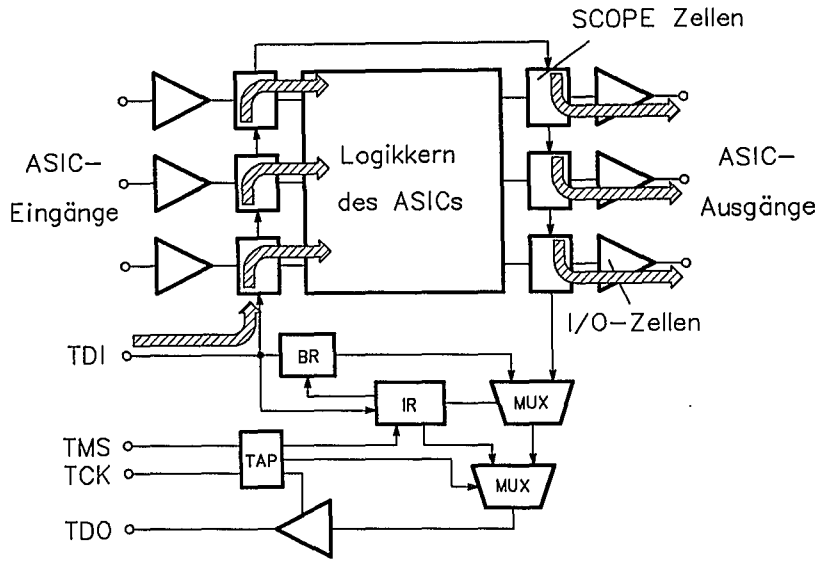


Bild 3.69: Boundary-Scan-Modus: Serielle Zuführung von Testdaten

Der Aufbau einer Boundary-Scan-Zelle ist in Bild 3.70 in einer von vielen Betriebsarten dargestellt. Das Bild zeigt einen Normalbetriebsszustand, in dem der Logikkern des ASICs transparent seine Daten von den Primäreingängen erhält bzw. seine Daten zu den Primärausgängen ausgeben kann. Parallel dazu wird die Scan-Kette über TDI-Register-TDO seriell geladen. Nach Abschluß des Ladens der Scan-Kette kann im Testmodus der Scan-Ketteninhalt an den Logikkern gelegt und seine Reaktion über ODI (*O*bservability *D*ata *I*n) in die Scan-Kette geladen und anschließend über TDO seriell auf den JTAG-Bus ausgegeben werden (siehe Bild 3.71). Die Steuerung der vielfältigen Betriebsarten erfolgt über den JTAG-Bus. Eine Test-Zugangsschnittstelle (TAP Test Access Port) steuert die Übernahme der Testdaten von TDI in ein seriell ladbares Befehlsregister (IR), in ein Bypass-Scan-Register (BR) oder in die eigentlichen Boundary Scan Zellen (SCOPE, siehe Bild 3.67 und 3.72).

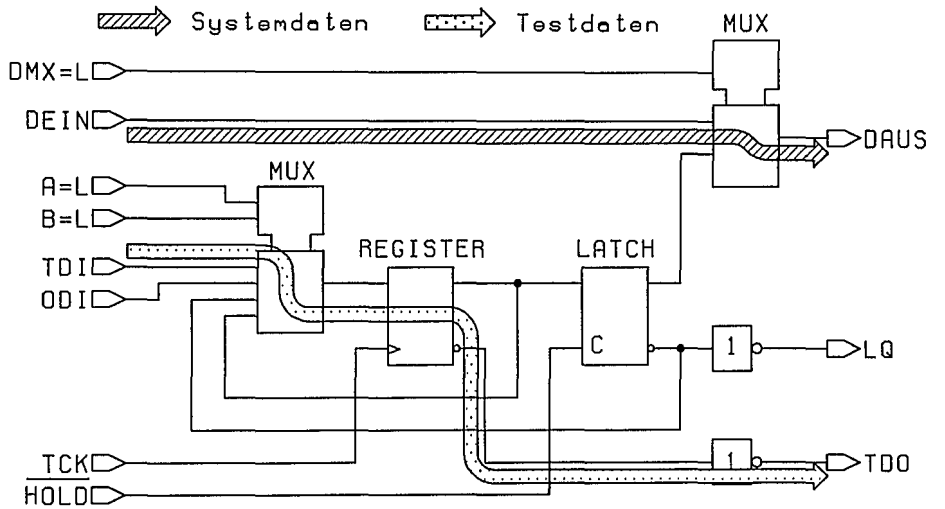


Bild 3.70: Aufbau einer Boundary-Scan-Zelle im Normalbetriebszustand und gleichzeitigem Laden der Scan-Kette

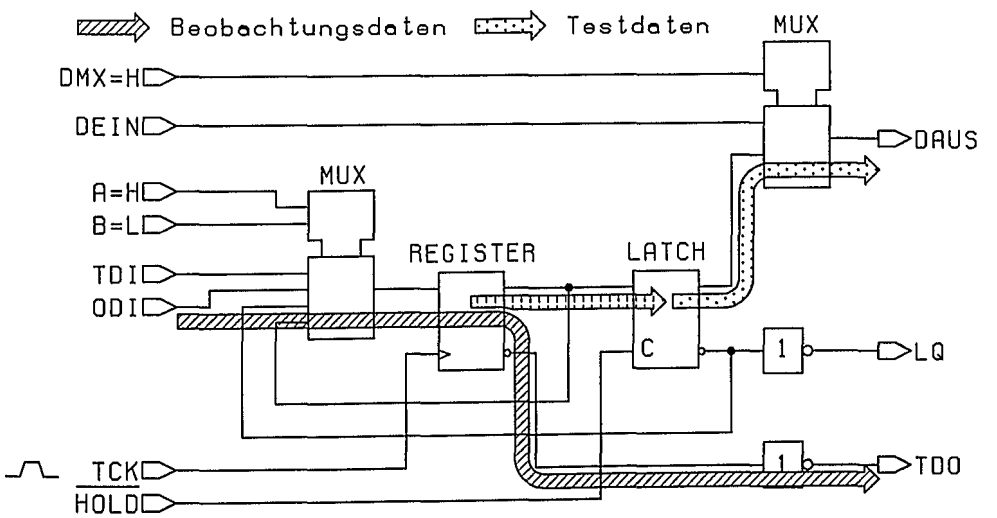


Bild 3.71: Testmodus des Boundary-Scan-Registers mit Ausgabe der Testdaten über DAUS und Übernahme der Beobachtungsdaten über ODI

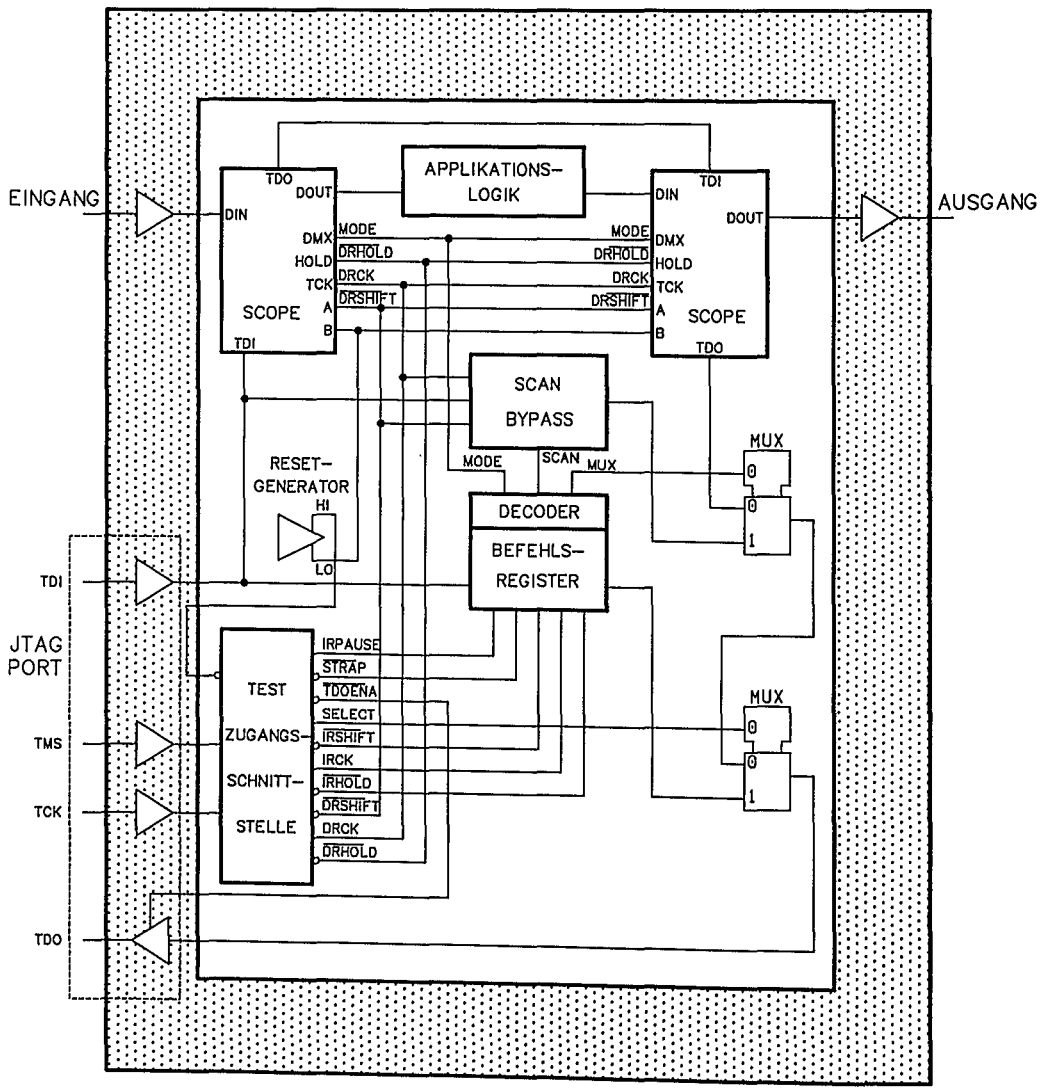


Bild 3.72: Steuerelemente für den Boundary-Scan-Test über den JTAG-Port in einem ASIC-Baustein

3.7 Selbsttest von Schaltungen (BIST)

Das Testen einer Schaltung erfordert die Anwendung einer Menge von Testvektoren auf die Schaltung und den Vergleich der aktuellen Schaltungsreaktion mit der korrekten Schaltungsreaktion. In den vorstehenden Abschnitten wurden Prinzipien vorgestellt, die davon ausgehen, daß ein Tester die Vektoren und erwarteten Reaktionen speichern, auf eine Schaltung anwenden und Tests auswerten kann. Da Tester sehr teuer sind, ist jeder solcher Test ein nicht vernachlässigbarer Kostenfaktor. Deshalb wurden Techniken entwickelt, einige der Funktionen des externen Testers auf jedem Chip selbst zu integrieren. Die Integration von zusätzlichen Schaltungsteilen auf dem Chip zur Erzeugung und zur Auswertung von Tests wird als BIST (*B*uilt *I*n Self *T*est) bezeichnet. Jede Testmethode benötigt eine Strategie für die Erzeugung von Testeingangssignalen, eine Strategie für die Gewinnung und Auswertung der Schaltungsreaktionen und eine Strategie für ihre effiziente Implementierung.

3.7.1 Testdatengenerierung

Im Prinzip ist es möglich, Testdaten manuell oder mit einem ATPG-Programm zu generieren und in einem ROM auf dem Chip abzulegen (off-line test generation). Da in aller Regel sehr große ROM nötig wären, werden bei BIST neue Testvektoren erzeugt, während andere angelegt werden (concurrent test pattern generation). Beim "random testing" wird die Schaltung mit zufällig erzeugten Testmustern beaufschlagt. Beim "exhaustive testing" werden alle möglichen Eingangsmuster an eine Schaltung angelegt.

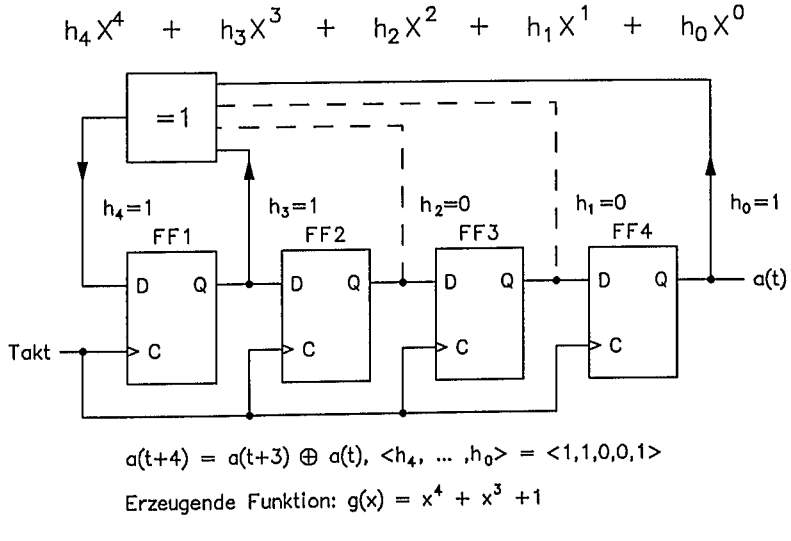


Bild 3.73: 4-stufiges LFSR zur Erzeugung von pseudozufälligen Testdaten

Zufällige Testmuster können durch linear rückgekoppelte Schieberegister (LFSR) einfach erzeugt werden. Ein Beispiel ist in Bild 3.73 dargestellt. Dort sind vier D-Flipflops in Serie geschaltet. Das Flipflop 1 wird durch Rückkopplung über das Exklusiv-Oder-Gatter versorgt. Die Wahl der Anschlüsse für die Rückkopplung definiert das erzeugende Polynom (hier: X^4+X^3+1). Wäre die Anzapfung statt bei Q von Flipflop 1 bei Flipflop 3, so wäre dem LFSR das erzeugende Polynom X^4+X+1 zugeordnet. Die Funktion eines LFSR kann als Modulo-2-Division des Schieberegisterinhalts durch das Koeffizientenmuster h_i des Rückkopplungspolynoms (erzeugendes Polynom) beschrieben werden. Bei einem Anfangsinhalt von 1000 im Schieberegister wird das Flipflop 1 sukzessive mit folgender Bitfolge geladen:

1 1 1 1 0 1 0 1 1 0 0 1 0 0 0 (1 1 1 1 0 1 ...)

(siehe Bild 3.74). Diese Folge wiederholt sich endlos. Da nach drei Takten der Inhalt von Flipflop 1 von Flipflop 4 übernommen ist, ist die Ausgangsbitfolge des LFSR identisch. Der Inhalt des LFSR durchläuft die hexadezimale Folge

8, C, E, F, 7, B, 5, A, D, 6, 3, 9, 4, 2, 1 (, 8, C, E, F, 7, B, 5 ...),

d.h. es werden alle Bitmuster mit Ausnahme von "0" in einer Pseudozufallsfolge periodisch wiederholt. Das benützte Polynom X^4+X^3+1 ist irreduzibel oder primitiv, das heißt, es kann selbst nicht als Produkt zweier anderer Polynome dargestellt werden. Ein irreduzibles Polynom vom Grad n erzeugt eine maximal lange Zufallsfolge der Länge (2^n-1) oder anders ausgedrückt, ein LFSR aus n Stufen erzeugt 2^n-1 unterschiedliche Zufallsmuster von n Bit mit Ausnahme des Musters aus lauter Nullen. Deshalb darf ein LFSR auch nicht mit 0 initialisiert werden. Eine Auswahl weiterer primitiver Polynome mit einer minimalen Anzahl von Anzapfungen (Rückkopplungen) ist in Tabelle 3.4 dargestellt.

Durch Modifikation des LFSR in Bild 3.73 kann auch 0000 als Muster zusätzlich erzeugt werden. Dazu müssen lediglich die invertierten Q-Ausgänge der Flipflops 1..3 über ein UND-Gatter dem XOR-Gatter zusätzlich zugeführt werden.

Tabelle 3.4: Eine Auswahl primitiver Polynome mit minimaler Zahl von Rückkopplungen

Grad	Minimal Primitives Polynom	Rückkopplung von Flip–Flop # gezählt von links nach rechts
2	X^2+X+1	1,2
3	X^3+X+1	1,3
4	X^4+X+1	3,4
5	X^5+X^2+1	3,5
6	X^6+X+1	5,6
7	X^7+X^3+1	4,7
8	$X^8+X^6+X^5+X^3+1$	2,3,5,8
9	X^9+X^4+1	5,9
10	$X^{10}+X^3+1$	7,10
11	$X^{11}+X^2+1$	9,11
12	$X^{12}+X^6+X^4+X+1$	6,8,11,12
13	$X^{13}+X^4+X^3+X+1$	9,10,12,13
14	$X^{14}+X^{10}+X^6+X+1$	4,8,13,14
15	$X^{15}+X+1$	14,15
16	$X^{16}+X^{12}+X^3+X+1$	4,13,15,16
17	$X^{17}+X^3+1$	14,17
18	$X^{18}+X^7+1$	11,18
19	$X^{19}+X^5+X^2+X+1$	14,17,18,19
20	$X^{20}+X^3+1$	17,20
21	$X^{21}+X^2+1$	19,21
22	$X^{22}+X+1$	21,22
23	$X^{23}+X^5+1$	18,23
24	$X^{24}+X^7+X^2+X+1$	17,22,23,24
25	$X^{25}+X^3+1$	22,25
26	$X^{26}+X^6+X^2+X+1$	20,24,25,26
27	$X^{27}+X^5+X^2+X+1$	22,25,26,27
28	$X^{28}+X^3+1$	25,28
29	$X^{29}+X^2+1$	27,29
30	$X^{30}+X^{23}+X^2+X+1$	7,28,29,30
31	$X^{31}+X^3+1$	28,31
32	$X^{32}+X^{22}+X^2+X+1$	10,30,31,32
33	$X^{33}+X^{13}+1$	20,33
34	$X^{34}+X^{27}+X^2+X+1$	7,32,33,34

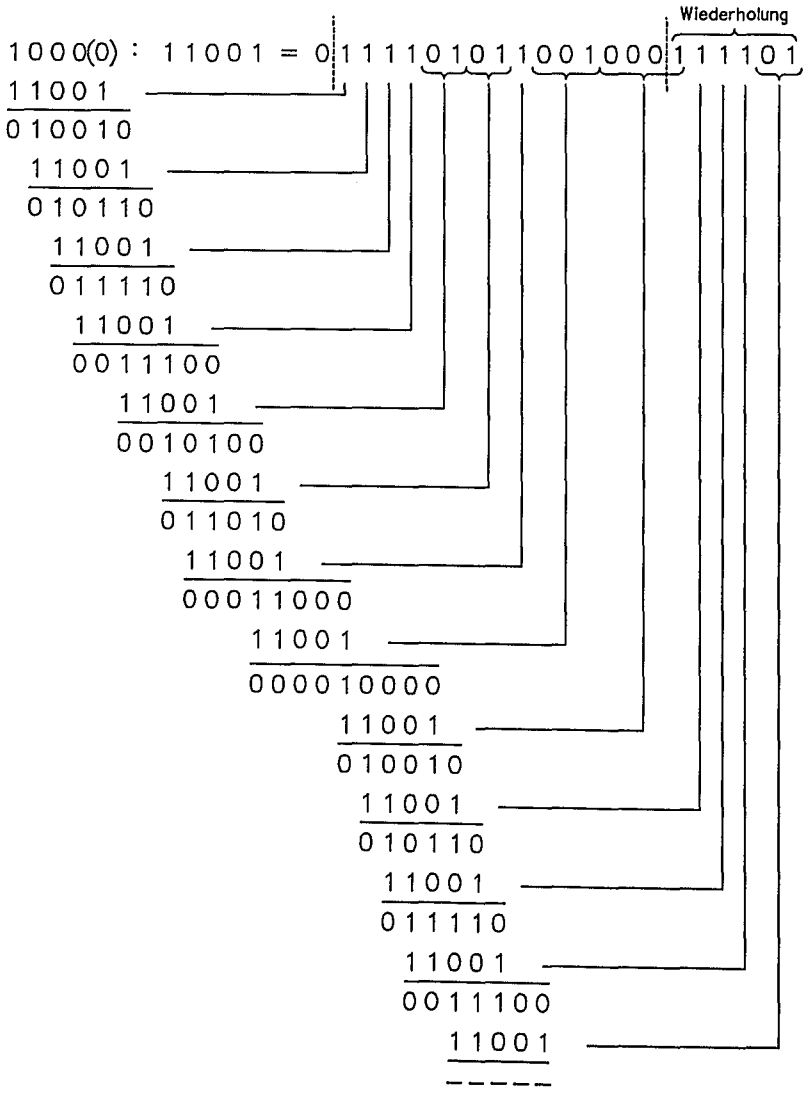


Bild 3.74: Modulo–2–Division des anfänglichen Registerinhalts 1000 durch das erzeugende Polynom X^4+X^3+1

3.7.2 Auswertung der Schaltungsreaktion (Pattern Compression)

Die Ablage eines Fehlerverzeichnisses auf einem Chip erfordert wie die Ablage eines Testvektorsatzes zu viel ROM-Speicherplatz. Deshalb werden Methoden zur Komprimierung der Testergebnisse angewandt. Die einzige Methode, die für BIST-Zwecke ernsthaft und mittlerweile mit schnell wachsender Verbreitung eingesetzt wird, ist die Signaturanalyse mit linear rückgekoppelten Schieberegistern (LFSR).

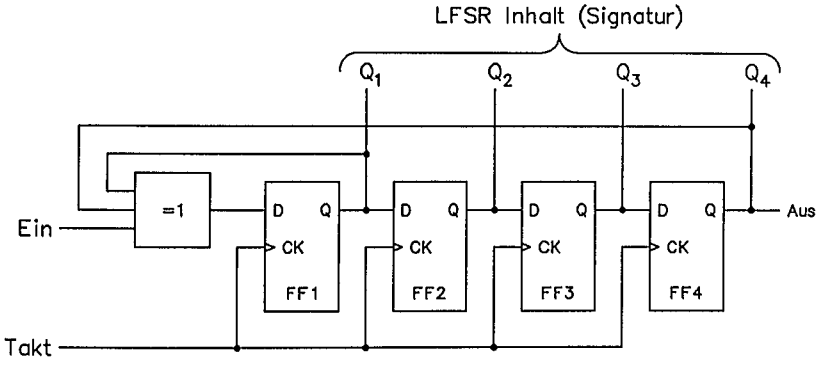


Bild 3.75: LFSR als serieller Signaturanalysator

Bild 3.75 zeigt das LFSR von Bild 3.73 modifiziert als seriellen Signaturanalysator oder "single input pattern compressor" SIPC. Der zu komprimierende Testdatenstrom wird bei "Ein" eingespeist und moduliert den Rückkopplungszweig des reinen LFSR.

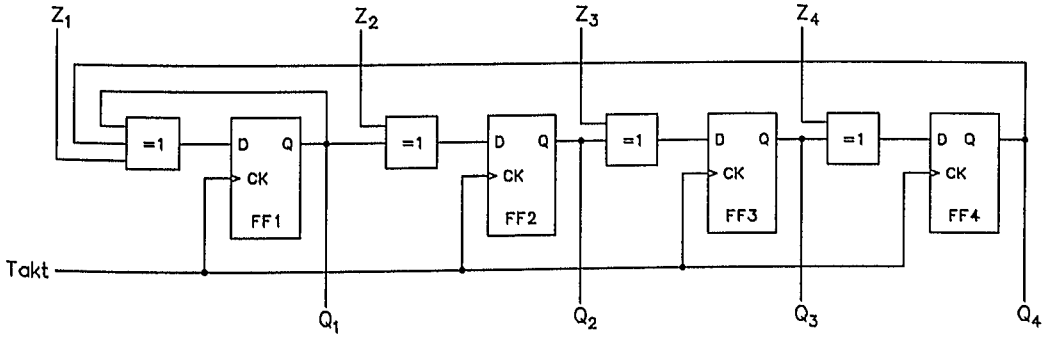


Bild 3.76: LFSR als paralleler Signaturanalysator bzw. Multiple-Input-Pattern-Compressor (MIPC)

Ein n-stufiges LFSR als SIPC kann maximal 2^n verschiedene Bitmuster enthalten. Wenn der Eingangsdatenstrom genügend lang ist, können manche Bitmuster wiederholt auftreten. Solange jedoch die Folge der Eingangsbit sich nicht selbst wiederholt, wiederholt sich in diesem Fall auch nicht die Musterreihenfolge im SIPC.

Der Inhalt des SIPC nach der Datenkompression ist die Signatur des Datenstroms. Es kann jedoch vorkommen, daß die Signatur eines fehlerhaften Datenstroms mit der des fehlerfreien Datenstroms übereinstimmt.

Das SIPC-Konzept kann leicht auf die Signaturanalyse paralleler Eingangssignale ausgedehnt werden. Dazu werden zwischen die Flipflops des LFSR in Bild 3.75 XOR-Gatter eingebracht und die parallel zu komprimierenden Signale über den zweiten XOR-Eingang eingespeist (MIPC: *M*ultiple *I*nput *P*attern *C*ompressor, siehe Bild 3.76).

3.7.3 Implementierung von BIST

Bild 3.77 a und b zeigen das Prinzip der Anwendung der Signaturanalyse. Die parallelen Signale können über einen Multiplexer seriell dem LFSR zugeführt werden, das als SIPC benützt wird. Nach serieller Übernahme aller Schaltungssignale $Z_1..Z_m$ kann die Signatur $Q_1..Q_r$ mit einer gespeicherten Signatur der korrekten Schaltung verglichen werden. Alternativ könnte die Signatur über den JTAG-Bus zur weiteren Auswertung ausgegeben werden. Bei Anwendung eines LFSR als MIPC können mehrere der Schaltungssignale $Z_1..Z_m$ gleichzeitig als $W_1..W_k$ dem MIPC zugeführt werden. Nach Übernahme aller Testsignale $Z_1..Z_m$ steht die Signatur $Q_1..Q_r$ zur weiteren Auswertung bereit.

Um die Signatur generieren zu können, muß der zu testende Schaltungsteil mit Testvektoren beaufschlagt werden. Im BIST-Fall erzeugt ein LFSR eine pseudozufällige Testvektorfolge, die über einen Multiplexer der zu testenden Schaltung (Schaltungsteil) zugeführt wird (siehe Bild 3.78). Die Ausgänge der Schaltung werden einem zweiten LFSR als SIPC oder MIPC zur Syndromgenerierung zugeführt. Dieses Prinzip ist für kombinatorische und sequentielle Schaltungen anwendbar.

Besonders elegant läßt sich die BIST-Methode in Verbindung mit einer Scan-Path-Struktur anwenden. Als Beispiel sei hier die in Abschnitt 3.6.4.4 besprochene Boundary-Scan-Technik herangezogen. Ein LFSR kann zunächst die Boundary-Scan-Zellen seriell mit Testdaten laden. Nach Durchlauf durch die zu testende Schaltung werden die Schaltungsausgangssignale mit Hilfe der Boundary-Scan-Zellen dem SIPC-LFSR zugeführt. Die nach einem Gesamttest entstandene Signatur wird schließlich mit der Soll-Signatur verglichen und so evtl. vorhandene Fehler festgestellt.

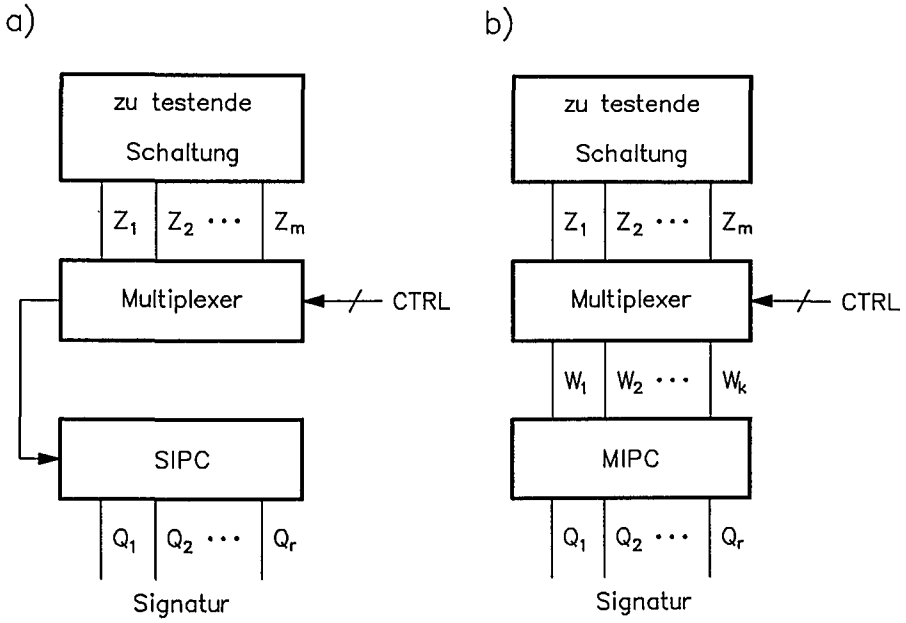


Bild 3.77: Komprimierung der Testergebnisse bei BIST; die Schaltungssignale werden einem LFSR als SIPC oder MIPC zur Kompression und Signalerzeugung zugeführt.

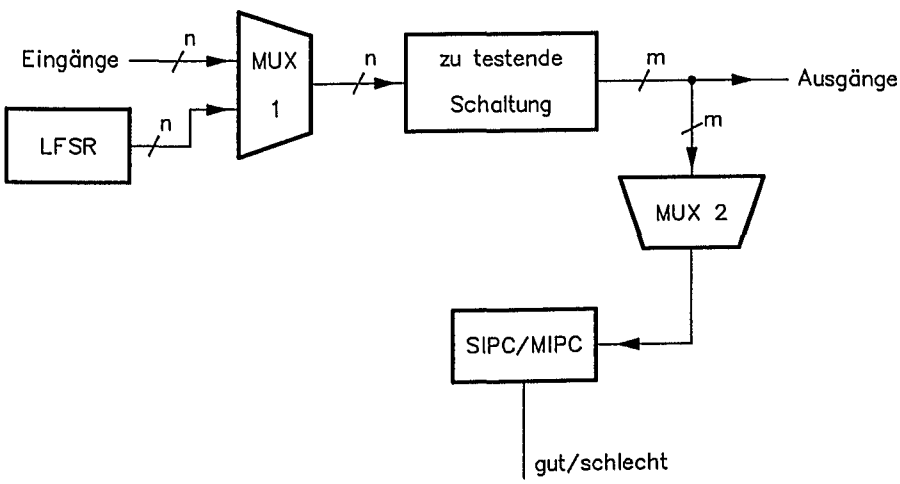


Bild 3.78: BIST-Struktur ohne Anwendung einer SCAN-Path-Struktur

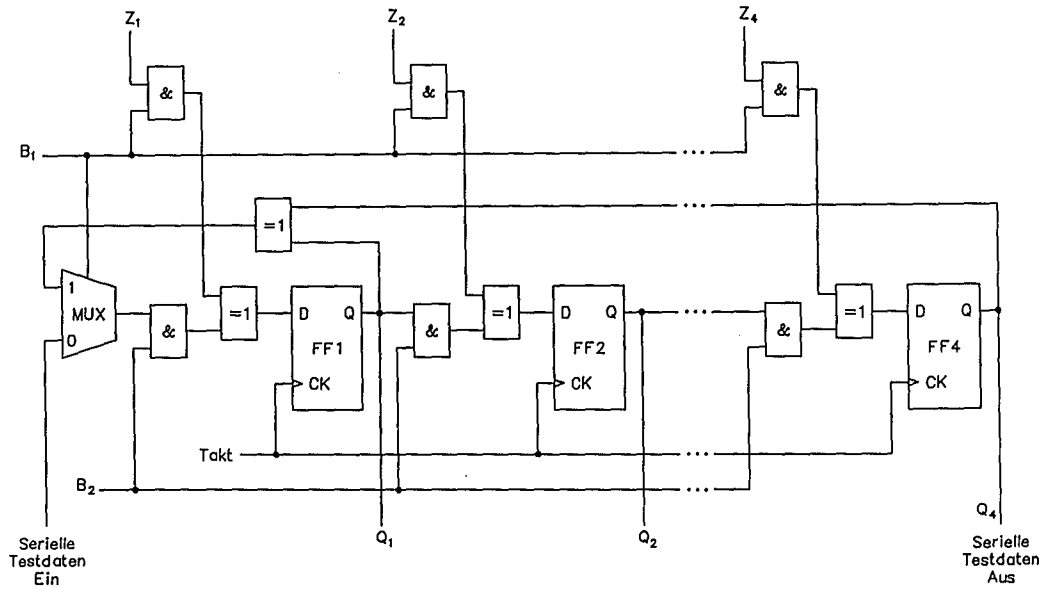


Bild 3.79: Register als BILBO (Built–In Logic Block Observer) rekonfigurierbar

Als letztes Beispiel sei das BILBO–Prinzip erwähnt (*B*uilt–*I*n *L*ogic *B*lock *O*bserver, siehe Bild 3.79). Dabei werden ohnehin in der Schaltung vorhandene Registerschaltungen rekonfiguriert. Wenn B1=1 und B2=0 ist, funktioniert das Register in Bild 3.79 als parallel ladbares Register. Wenn B1=0 und B2=1 ist, wirkt das Register als seriell ladbares Schieberegister. Testdaten können seriell ein– und ausgelesen werden. Durch B1=1 und B2=1 wird das Register zu einem MIPC bzw. parallelen Signaturanalysator oder, falls die Eingänge Zi konstantgehalten werden, zu einem LFSR–Testdatengenerator. Das Register wird durch B1=0 und B2=0 zurückgesetzt.

Literatur zu Kapitel 3

/1/ A. Mukherjee: Introduction to NMOS and CMOS VLSI Systems Design. Prentice Hall, 1986

/2/ E. S. Young: Fundamentals of Semiconductor Devices. McGraw Hill, 1979

/3/ L. Herman: Controlling CMOS Latch-up. VLSI Design, April 1985

/4/ H.-U. Post: Entwurf und Technologie hochintegrierter Schaltungen. Teubner-Verlag, Stuttgart, 1989

/5/ H. K. Reghbati: Tutorial: VLSI Testing & Validation Techniques. IEEE Computer Society, ISBN 0-8186-0668-1

/6/ C. C. Timoc: Selected Reprints on Logic Design for Testability. IEEE Computer Society, ISBN 0-8186-0573-1

/7/ E. J. McCluskey: Testing Semi-Custom Logic. Semiconductor International, Sept. 1985, S. 118 ff.

/8/ M. C. Markowitz: High Density ICs need design-for-test methods. EDN 24.NOV.1988, S. 73 ff.

/9/ Texas Instruments: TSC 500 Series Software Macros Manual. März 1989

/10/ Texas Instruments: 1 MIKRON G/A TGC100 DESIGN MANUAL. April 1988

/11/ F. F. Tsui: LSI/VLSI Testability Design. Mc Graw Hill 1987, ISBN 0-07-065341-0

/12/ R. J. Fengate: Introduction to VLSI-Testing.

/13/ G. Rabbat: Handbook of Advanced Semiconductor Technology and Computer Systems.

/14/ D. P. Siewiorek, R. S. Swarz: The Theory and Practice of Reliable Systems Design. DIGITAL Press, ISBN 0-932376-13-4

/15/ Austria Mikro Systeme (AMS): 2-Micron Gate Array Databook.

/16/ P. Ammon: Gate Arrays. Heidelberg Hüthig 1986, ISBN 3-7785-1063-0

/17/ LSI-Logic Corporation: CMOS Macrocell Manual. Juli 1985

Fragen zum 3.Kapitel

F3.1 Geben Sie die Übertragungskennlinie und die Stromaufnahme eines CMOS-Inverters an!

F3.2 Wodurch unterscheiden sich "channeled" Gate-Arrays von "Sea-of-Gates" Gate-Arrays und wo werden Sea-of-Gates vorteilhaft eingesetzt?

F3.3 Was ist eine Poly-Silizium-Leitung und welche Eigenschaften hat diese?

F3.4 Warum muß bei einem CMOS-Gatter der p-Kanal-Transistor breiter ausgelegt werden?

F3.5 Skizzieren Sie $R_{on}=f(U_1)$ eines durchgeschalteten CMOS-Transfer-Gates!

F3.6 Beschreiben Sie das Schaltverhalten eines CMOS-Inverters beim Übergang von "0" auf "1" am Ausgang! Geben Sie ein einfaches Ersatzmodell in den jeweiligen Betriebszuständen an!

F3.7 Nennen Sie die Einflußgrößen auf die Gatterlaufzeit einer CMOS-Technologie!

F3.8 In welchem Bereich liegt die Schwankungsbreite der Signallaufzeiten bei CMOS-Gattern?

F3.9 Was versteht man unter Vorbereitungszeit und Nachhaltezeit bei taktflankengetriggerten Flipflops und wie kommen diese Zeiten zustande?

F3.10 Wonach richtet sich die maximale Taktfrequenz einer Logikfunktion?

F3.11 Wodurch kommen SPIKES zustande?

F3.12 Was versteht man unter RACES in einer Logikschaltung?

F3.13 Nennen Sie geeignete Maßnahmen, um die Kontrollierbarkeit innerer Netze zu verbessern!

F3.14 Welche Maßnahmen sind geeignet, um Logikblöcke zu entkoppeln?

F3.15 Wie können lange Zählerketten zur Verbesserung der Testbarkeit aufgebrochen werden?

F3.16 Was ist hinsichtlich der Initialisierung von Zustandsautomaten zu beachten?

F3.17 Auf welche Weise können intern Testsignale erzeugt werden?

F3.18 Welche Bedeutung haben Multiplexer zur Verbesserung der Testbarkeit?

F3.19 Was ist ein Scan–Path? Beschreiben Sie ein Scan–Path–Konzept am Beispiel eines Zustandsautomaten!

F3.20 Was versteht man unter den Begriffen "Primäreingang" und "Primärausgang"?

F3.21 Erklären Sie die Begriffe "Sensibilisierung", "Konsistenz" und "Untestbarkeit".

F3.22 Erstellen Sie den D–Ausbreitungswürfel für ein Exklusiv–Oder–Gatter.

F3.23 Ermitteln Sie mit Hilfe des D–Algorithmus einen Test für den Fehler 7/1 in Schaltung Bild 3.40.

F3.24 Diskutieren Sie die Probleme, Vor– und Nachteile des funktionellen Tests digitaler Schaltungen.

F3.25 Erstellen Sie einen Regelsatz für die Ermittlung der Kontrollierbarkeit und Beobachtbarkeit eines NAND–Gatters mit drei Eingängen nach dem SCOAP–Verfahren.

F3.26 Entwickeln Sie Testvektoren für die Fehler 6/0 und 6/1 von Schaltung Bild 3.43.

F3.27 Erklären Sie die Testbarkeitsmaße "Kontrollierbarkeit" und "Beobachtbarkeit". Wozu sind solche Informationen nützlich?

F3.28 Konstruieren Sie aus einem einfachen D–Flipflop, NAND–Gattern und Invertern ein SCAN–FLIPFLOP.

F3.29 Diskutieren Sie Anwendungsmöglichkeiten des JTAG–Testkonzepts für Baustein– und Systemtests.

F3.30 Das LFSR von Bild 3.73 soll zu einem SPIC erweitert werden. Ermitteln Sie die Signatur des Bitstroms 111101011001, wenn das LFSR anfangs mit 0 vorgeladen ist.

F3.31 Gegeben sei die Schaltung von Bild 3.45. Ermitteln Sie SCOAP-Testbarkeitsmaße dieser Schaltung.

F3.32 Warum ist die Schaltung von Bild 3.53 untestbar?

F3.33 Erstellen Sie Testvektoren für den Test auf Stuck-at-0/1 der Schaltung nach Bild 3.54.

F3.34 Erklären Sie die Funktionsweise des LSSD-Testverfahrens.

F3.35 Gegeben sei die Chip-Testarchitektur von Bild 3.72. Ergänzen Sie diese Architektur um BIST mit einem seriellen Testpatterngenerator und einem seriellen Signaturanalysator.

4. Entwurfswerkzeuge auf CAE-Workstations

Zur Hardware: Seit dem Aufkommen leistungsfähiger Workstations (z. B. Apollo; Sun; MicroVAX) mit hochauflösenden Farbgrafik-Bildschirmen etwa Mitte der 80er Jahre sind CAE-Entwurfswerkzeuge an Arbeitsplatzrechnern verfügbar. Eine derartige CAE-Workstation hat einen 32-Bit-Mikrorechner, mindestens 8 MByte Speicherausbau, einen Massenspeicher mit mindestens 150 MByte und einen 19-Zoll-Farbgrafikbildschirm mit mindestens 1000x800 Bildpunkten Auflösung. Derzeit liegt die CPU-Leistung mit Standard-Prozessoren bei ca. 4 MIPS. Über ein lokales Netz (z. B. Token-Ring; Ethernet) können mehrere Workstations miteinander verbunden werden. Damit können die Entwurfswerkzeuge und Bibliotheken im Netz verteilt werden. Bild 4.1 zeigt eine Anlagenkonfiguration mit 5 CAE-Workstations vernetzt über einen Token-Ring. Über Ethernet kann diese Gruppe mit weiteren Workstations verbunden sein. Darüberhinaus läßt sich die skizzierte Anlagenkonfiguration lokal über Ethernet oder über eine Fernleitung (z. B. DATEX-P) mit einem Mainframe-Rechner verbinden. Für die hier vorliegenden Aufgaben ist der Personalcomputer nicht ausreichend. Aus diesem Grunde wurden Personalcomputer hochgerüstet, damit erreichen sie annähernd die Leistungsfähigkeit von Workstations. Ein "high-end"-Personalcomputer liegt aber nahezu in die Preisklasse einer "low-end"-Workstation. Die Ausbaufähigkeit spricht jedoch eindeutig für die Workstation. Personalcomputer werden im Rahmen eines integrierten Entwurfssystems nur sehr bedingt für CAE-Aufgaben eingesetzt.

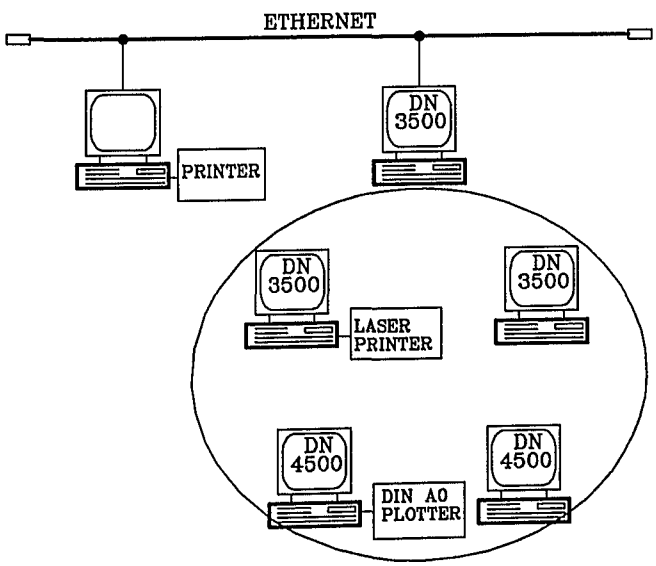


Bild 4.1: Beispiel einer CAE-Workstation-Hardwarekonfiguration

Zur Software: Die Entwurfswerkzeuge haben sich ähnlich wie die Hardware in den letzten Jahren sehr rasch fortentwickelt. Seit geraumer Zeit gibt es Circuit–Simulatoren, Logik–Simulatoren und Layout–Programmpakete als Einzelwerkzeuge an Mainframe–Rechnern. Integrierte Gesamtsysteme sind seit ca. Mitte der 80er Jahre auf CAE–Workstations verfügbar. In Bild 4.2 ist skizziert, was man unter einem integrierten Gesamtsystem für CAE–Aufgaben versteht. Wesentliches Merkmal ist insbesondere die gemeinsame Datenbasis (z. B. Design–Files, Bauteil–Bibliotheken, prozeßspezifische Bibliotheken), so daß die in einem Entwurfsschritt generierten Daten unmittelbar für die nächsten Entwurfsschritte verfügbar sind.

Ziel der nachstehenden Erläuterungen ist es, ein Grundverständnis für die Entwurfswerkzeuge aufzubereiten, um so schneller zu einem tieferen Verständnis zu kommen. Es soll bewußt hier kein "Operator–Wissen" aufbereitet werden. Soweit dies zum Verständnis notwendig ist, orientieren sich die Beispiele an der IDEA–Software von Mentor–Graphics.

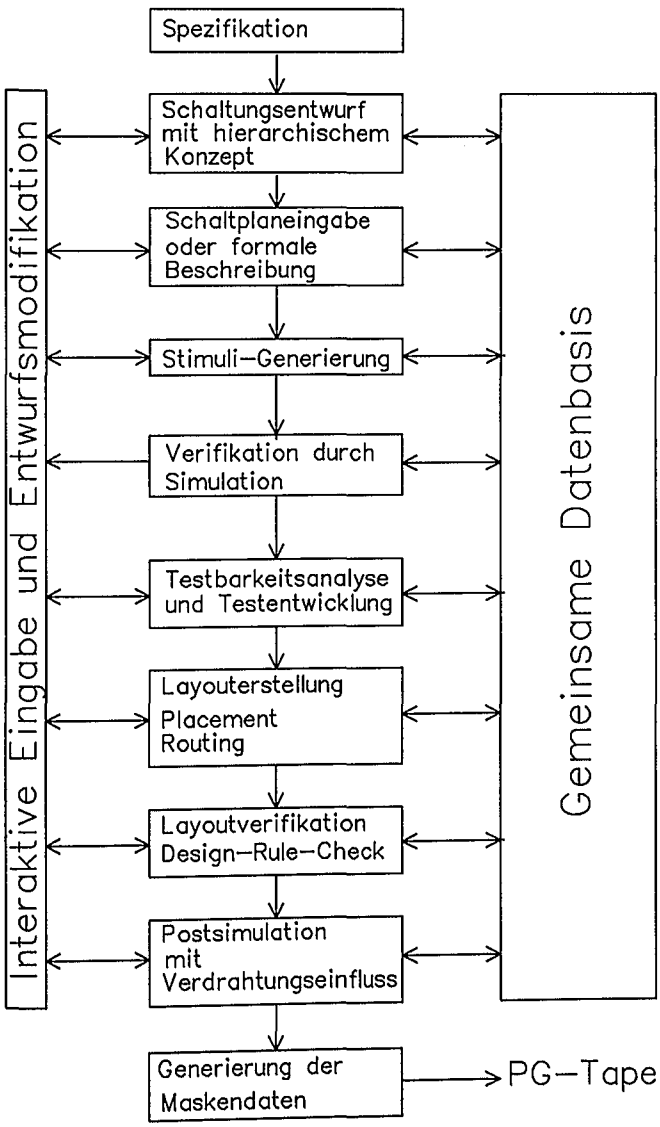


Bild 4.2: Designablauf beim Entwurf Integrierter Schaltungen

4.1 Grafische Schaltplaneingabe

Nach dem Schaltungsentwurf wird die nunmehr vorliegende Schaltungsidee durch eine die Spezifikationsvorgaben erfüllende Schaltungsstruktur mit Hilfe der grafischen Schaltplaneingabe festgelegt. Aus Gründen der Übersichtlichkeit, des besseren Verständnisses und der Überprüfbarkeit wird der Entwurf hierarchisch strukturiert. In der obersten Ebene werden im allgemeinen die Blockfunktionen festgelegt. In den darunter liegenden Ebenen erfolgt die weitere Konkretisierung der Schaltung. Damit wird insbesondere auch die Funktionalität veranschaulicht. Bild 4.3 zeigt als Beispiel den Bildschirmaufbau bei der Schaltplaneingabe mit NETED (Firma Mentor–Graphics).

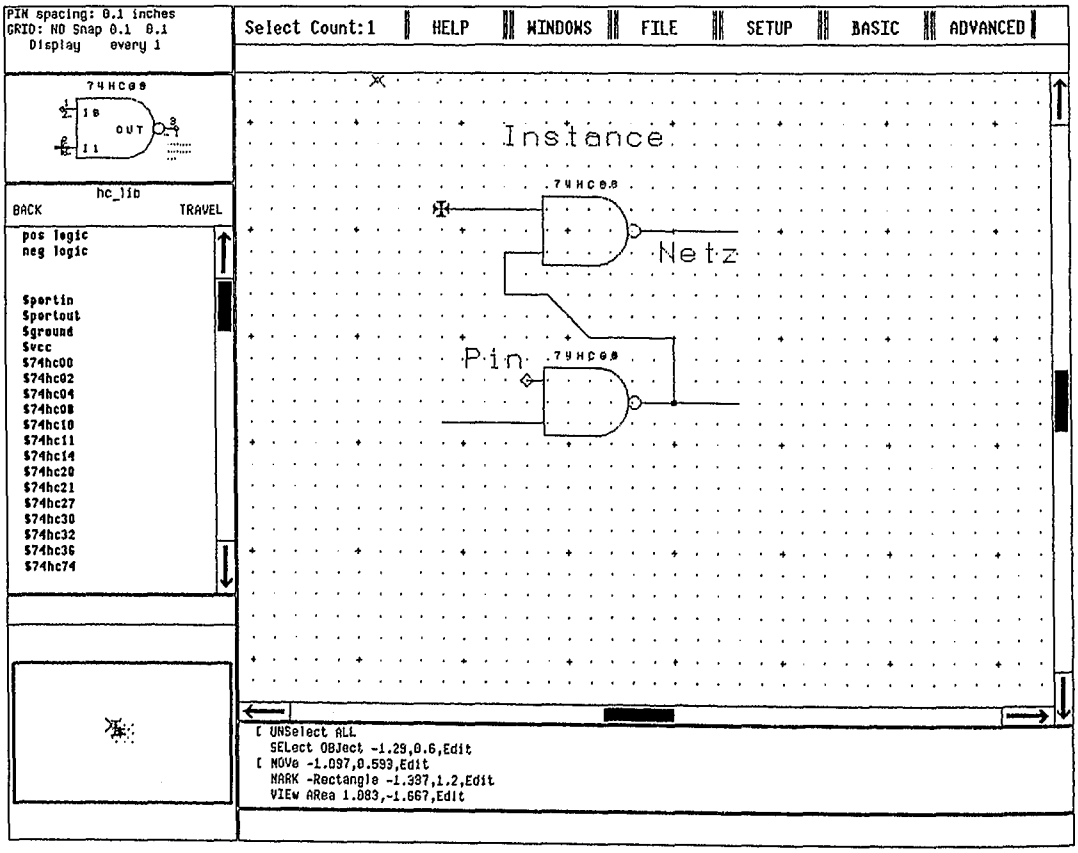


Bild 4.3: Beispiel des Bildschirmaufbaus bei der graphischen Schaltplaneingabe (NETED™ : Mentor–Graphics)

Die grafische Schaltplaneingabe enthält als Kern ein Edit-Window, in dem die Schaltungsstruktur aufbereitet wird. Über ein weiteres Fenster kann eine Symbolbibliothek oder Makrozellen-Bibliothek angewählt werden. Aus der Symbolbibliothek lassen sich die Schaltungssymbole entnehmen. Sie können unmittelbar ins Edit-Window plaziert werden. Im Edit-Window erfolgt dann der Aufbau des Schaltplans durch Verbindung der Symbole an deren Anschlußpins; es sind die "Netze" des Schaltplans festzulegen. Für die unterschiedlichen Realisierungstechniken von Schaltungen gibt es jeweils eine vorbereitete Symbolbibliothek für verfügbare Makrozellen.

Der Schaltplan enthält folgende drei Grundelemente:

- Instances: Das sind Symbole für Schaltungselemente, sie sind aus der gewählten Symbolbibliothek (in Bild 4.3: HC-LIB im mittleren Fenster links) zu entnehmen.

- Pins: Das sind die Anschlußpins, die bei dem jeweiligen Symbol nach außen führen.

- Nets/Subnets: Das sind "Netze" für die Verbindung der Pins von Symbolen bzw. die "Netze" für Hierarchieverbindungen, oder z. B. die Verbindungsbezeichner für nachgeordnete Busabzweige. Jedes Netz besteht aus einem oder mehreren Netzsegmenten.

Alle diese Grundelemente lassen sich in der Anordnung zueinander editieren. Teilbereiche können u.a. selektiert, verschoben, kopiert, gelöscht, gespiegelt, gedreht werden. Die Steuerung dieser Editiervorgänge erfolgt über Pop-Up-Menüs oder durch Kommandoeingabe. Zur Kennzeichnung von Bauteilen, von Netzen und Pins, sowie zur Festlegung von bestimmten Eigenschaften und Merkmalen, können an jedem Instance/Pin/Net/Subnet zusätzliche Bezeichner (Properties) hinzugefügt werden. Bestimmte Properties sind für nachgeordnete Entwurfswerkzeuge zur Festlegung von bestimmten Eigenschaften und Merkmalen erforderlich. So benötigt z. B. der Logiksimulator u. a. folgende Festlegungen an einem Pin: Anstiegszeit, Abfallzeit, Verzögerungszeit, Treiberstärke.

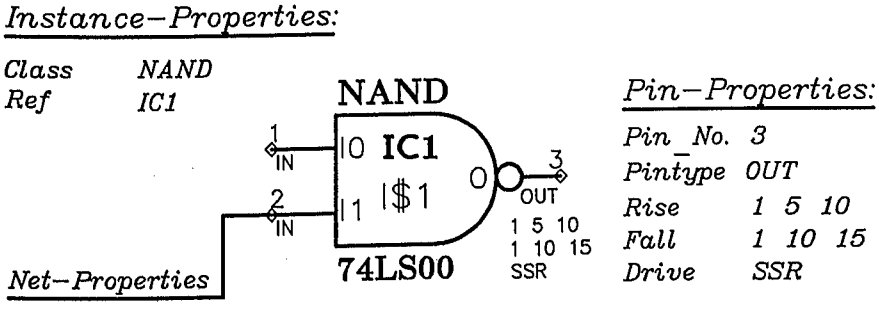


Bild 4.4: Zur Anfügung von Properties

Diese Bezeichner (Properties) zur Festlegung gewisser Merkmale und Eigenschaften sind im allgemeinen Variable, die einen bestimmten Wert annehmen können. Über solche Properties können sehr flexibel Daten für nachgeordnete Entwurfswerkzeuge an eine gegebene Schaltungsstruktur angefügt werden. Einige Beispiele von Properties bei der grafischen Schaltplaneingabe mit NETED™ sind:

- Properties am Symbol, u. a.:

Name	Wert	Bedeutung
INST	I$2	Symbolbezeichner für internen Gebrauch (Handle–Name),
REF	IC3	Symbolbezeichner für den Anwender,
COMP	74HC04	Zuordnung zum Bauteil,
MODEL	$nand	Festlegung des Modells für den Logik–Simulator,
MODEL	R	Festlegung des Modells für den Circuit–Simulator,
SPICEPAR	220	Wert für Modellparameter.

- Properties am Pin, u.a.:

Name	Wert	Bedeutung
PIN	I0	Pinbezeichner,
PIN_NO	3	Pinnummer–Zuordnung,
PINTYPE	OUT	Pintyp–Festlegung,
RISE	1 5 10	Anstiegszeit (Min. Typ. Max.),
FALL	1 10 15	Abfallzeit (Min. Typ. Max.),
DRIVE	SSR	Treiberstärke

- Properties am Net, u. a.:

Name	Wert	Bedeutung
INIT	...	Festlegung des Anfangszustandes,
DTIME	...	Festlegung einer Verzögerungszeit.

Bild 4.4 verdeutlicht das Hinzufügen von Properties. Jeder Bezeichner hat einen Eigner (Symbol/Pin/Net/Subnet). Für die Änderung eines Bezeichners muß zuerst der Eigner selektiert werden.

Ergebnis der grafischen Schaltplaneingabe ist ein Design-Directory mit Files, die alle notwendigen Daten für nachgeordnete Entwurfwerkzeuge enthalten. Aus den Design-Files kann eine Netz- und Verbindungsliste erzeugt werden. Vor einer Weiterverarbeitung in einem nachgeordneten Entwurfswerkzeug müssen in einem EXPAND-Lauf die für dieses Entwurfswerkzeug relevanten Properties extrahiert werden. Bei einer objektorientierten Datenbasis ist das "Extrahieren" der für ein bestimmtes Entwurfswerkzeug erforderlichen Merkmale und Eigenschaften nicht notwendig; vielmehr kann direkt auf die in der Datenbasis abgelegten Eigenschaften und Merkmale zugegriffen werden.

Für die Steuerung der grafischen Schaltplaneingabe steht eine Makro-Kommandosprache zur Verfügung. Beispielsweise wird mit ACTIVATE COMPONENT <Pfad>/$74HC00 das Symbol für das NAND-Gatter 7400 aus der ausgewählten HC-LIB ins EDIT-Window übernommen und an die Stelle plaziert, auf der der Grafik-Cursor positioniert ist. Die gleiche Aktion ist über ein Pop-Up-Menü zu steuern. Bild 4.5 zeigt den Schaltplan eines hierarchischen Schaltungsentwurfs. Der Funktionsblock "Ablaufsteuerung" wird z.B. in einer tiefer liegenden Hierarchiestufe weiter konkretisiert.

Einige Grundfunktionen für die Editierung eines Schaltplans sind:

- Aktivieren: von Symbolen aus der Symbolbibliothek (PARTS),
- Selektieren: von Bereichen, Symbolen, Netzen, Pins, u.a. (SELECT, GROUP),
- Kopieren oder Verschieben: von selektierten Bereichen (COPY, MOVE),
- Verbinden: von Pins durch Netze (DRAW NET),
- Festlegung: von Variablen (PROPERTIES),
- Löschen: von selektierten Bereichen (DELETE).

Ein TRANSSCRIPT-Window dient zur Protokollierung des Ablaufs und für Fehlerhinweise.

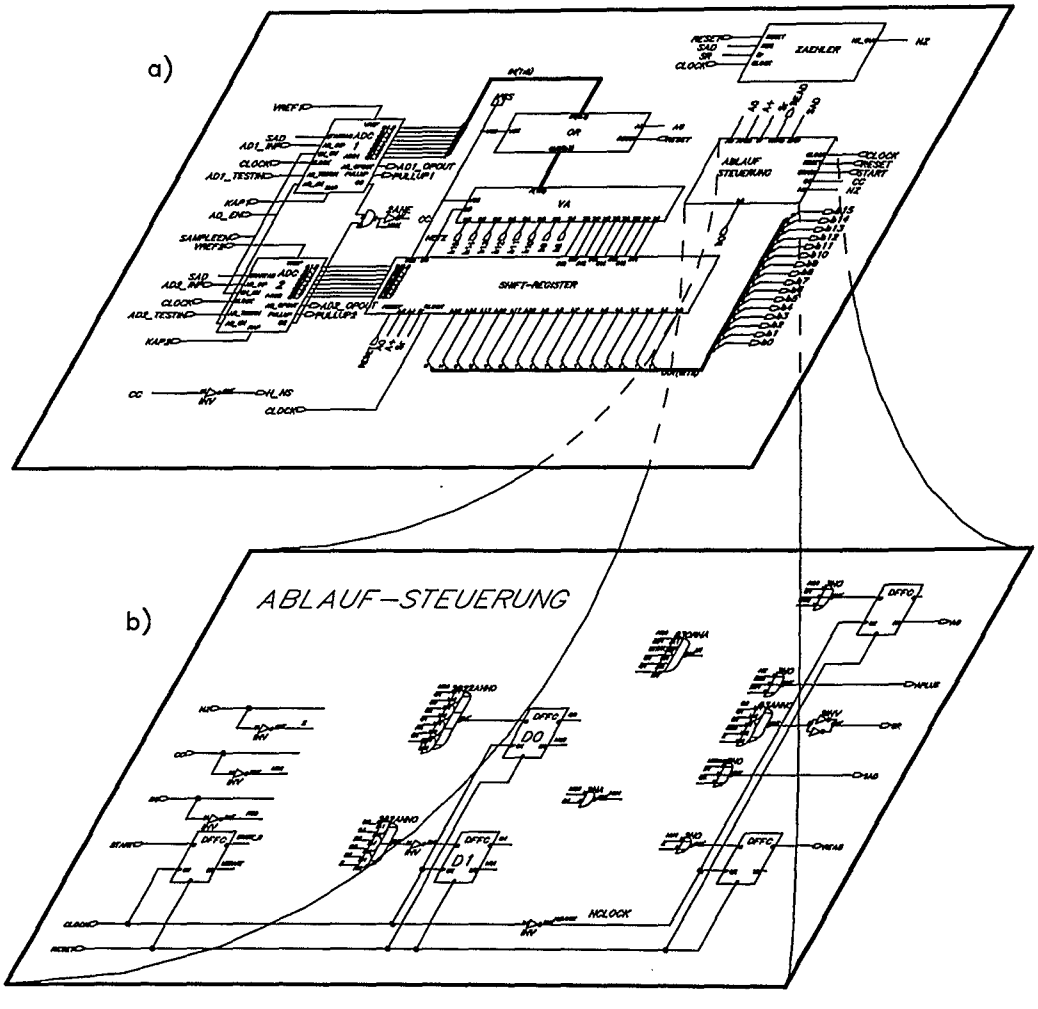


Bild 4.5: Beispiel einer hierarchischen Schaltplaneingabe a) Blockstruktur b) Teilfunktion: Ablaufsteuerung

Weitere Gruppen von Grundfunktionen für die Bearbeitung der Schaltplaneingabe z.B. mit NETED™ sind:

WINDOWS: Festlegung des Bildschirmaufbaus; u. a. Größe und Aufteilung von Fenstern.

FILE: Festlegung des Zugriffs auf Files and Ausgabemedien (z. B. Plotter); weiterhin ist z. B. mit OPEN SHEET BELOW der Einstieg in die nächst untere Hierarchiestufe möglich. Dazu muß vorher der Block, in den eingestiegen werden soll, selektiert werden.

SETUP: Festlegung von Vorbesetzungen (z. B. für Blattgröße, Raster, Einrasten, Schriftart und Schriftform für Texteingaben).

BASIC: Hier sind einige Grundfunktionen (z. B. Konsistenzprüfung) und Funktionen zum Spiegeln und Drehen von selektierten Bereichen zusammengefaßt.

ADVANCED: In dieser Gruppe sind Sonderfunktionen zusammengefaßt. Mit STATUS können z. B. alle Daten (u.a. alle Properties) die an einem selektierten Symbol/Netz hängen, dargestellt werden. Mit ADD BLOCK wird ein neues Blocksymbol erstellt. Nach Anfügen der Netze für abgehende Anschlüsse an das neue Blocksymbol werden mit CONNECT BLOCK die Anschluß-Pins des Blocksymbols erzeugt. Mit SAVE wird das neue Blocksymbol generiert und zur weiteren Verwendung abgespeichert.

Neue Symbole lassen sich insbesondere auch mit dem Entwurfswerkzeug SYMED™ erstellen. SYMED™ ist ein eigenständiges Werkzeug für die Symbolerzeugung. Die Funktion ADD BLOCK / CONNECT BLOCK ist nur ein Hilfsmittel, um von NETED™ aus auch einfache Rechteck-Blöcke bzw. Symbole zu erzeugen. Mit dem SYMED™ könnte beispielsweise das bereits im NETED™ durch ADD BLOCK erzeugte Symbol weiter editiert werden. Vom Schaltplan und von den erstellten Blocksymbolen wird die Versions-Nummer bei einer Überarbeitung fortgeschrieben. Um sicherzustellen, daß die neueste Version eines Blocksymbols verwendet wird, muß ein UPDATE PART vorgenommen werden.

Schließlich ist vor dem Verlassen des NETED™ ein CHECK SHEET vorzunehmen. Hier werden einige Vollständigkeits- und Konsistenzprüfungen durchgeführt. Bei Fehlern wird in geeigneter Weise darauf hingewiesen.

4.2 Schaltungsverifikation durch Simulation

Wie bereits dargelegt, ist bei integrierten Schaltkreistechniken die Schaltungsverifikation durch Simulation unverzichtbar. Ein diskreter Aufbau berücksichtigt nicht die Realität auf Silizium. Jeder Technologiedurchlauf ist außerordentlich kostspielig. Um Technologiedurchläufe auf das notwendige Maß zu beschränken, muß vorher das Schaltungsverhalten sehr weitgehend simuliert und verifiziert werden. Simulation bedeutet Analyse und Modifikation am Modell. Die Genauigkeit der Schaltungsverifikation durch Simulation hängt vom Analyseverfahren und von der Modellgenauigkeit der verwendeten Elemente ab. Bei genaueren Analyseverfahren werden genauere Modelle benötigt. In Bild 4.6 ist der Zusammenhang zwischen Schaltungskomplexität, Analyseverfahren und Modellgenauigkeit veranschaulicht.

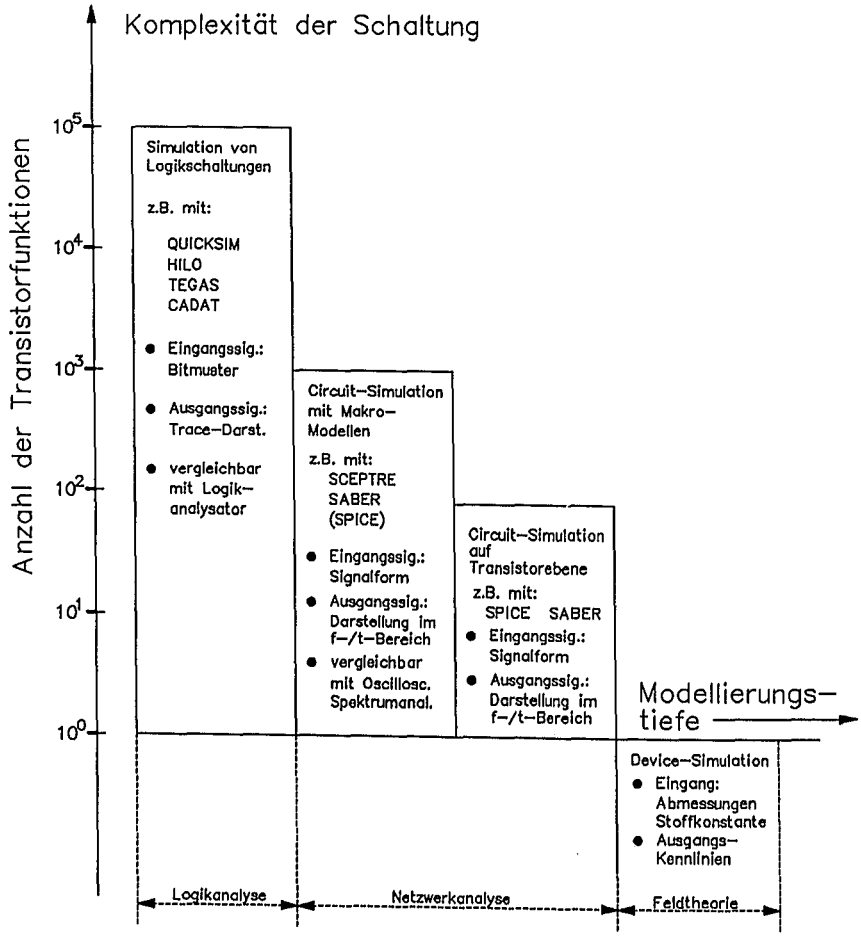


Bild 4.6: Zur Einteilung der Werkzeuge zur Schaltungsverifikation

Für die Schaltungssimulation gibt es heute sehr leistungsfähige Simulationswerkzeuge; sie lassen sich in folgende Gruppen einteilen:

- Device–Simulation: Damit können Einzelhalbleiter und geometrische Strukturen simuliert werden. Bei vorgegebener Geometrie und vorgegebenen Halbleitereigenschaften (Dotierungsprofile u.a.) lassen sich mit Methoden der Feldtheorie Kennlinien und Modellparameter ermitteln.

- Circuit–Simulation: Hier werden auf Transistorebene und Makro–Modellebene Schaltkreise zeit– und frequenzkontinuierlich simuliert. Unter Zugrundelegung geeigneter Modelle für die Schaltungselemente (Bild 4.7a) wird der Schaltkreis unter Berücksichtigung der Erregung durch Eingangssignale mit Methoden der Netzwerkanalyse berechnet. Als Ergebnis erhält man die kontinuierlichen Verläufe der Spannungen an den Netzen (Knotenpotentiale) und die Verläufe der Zweigströme jeweils im Frequenz– bzw. im Zeitbereich.

- Logik–Simulation: Mit vereinfachten Modellen auf Logikebene (Bild 4.7b) werden die Logikzustände (0–1–X) von Netzen für gegebene Eingangssignalzustände einer Logikschaltung berechnet.

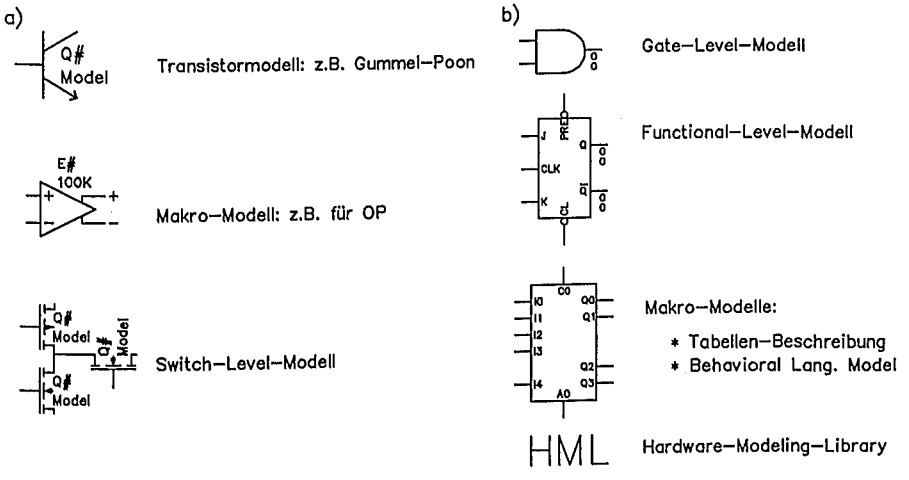


Bild 4.7: Übersicht der verschiedenen Modellierungsebenen von Schaltungselementen
a) Analog–Modelle
b) Digital–Modelle

Zur Circuit–Simulation: Mit der aufwendigen Circuit–Simulation können nur kleinere Schaltungsteile verifiziert werden. Bei komplexen Digitalschaltungen beschränkt sich die Circuit–Simulation auf die Charakterisierung von Zellen und deren Verbindungsstruktur. In Bild 4.8 ist ein Beispiel für die Circuit–Simulation mit MSPICE™ von Mentor–Graphics dargestellt. Es handelt sich um eine einfache nichtlineare Schaltungskonfiguration, die auf Transistorebene im Zeitbereich simuliert wird. In den Blöcken befindet sich ein Inverter–Schaltkreis auf Transistorebene.

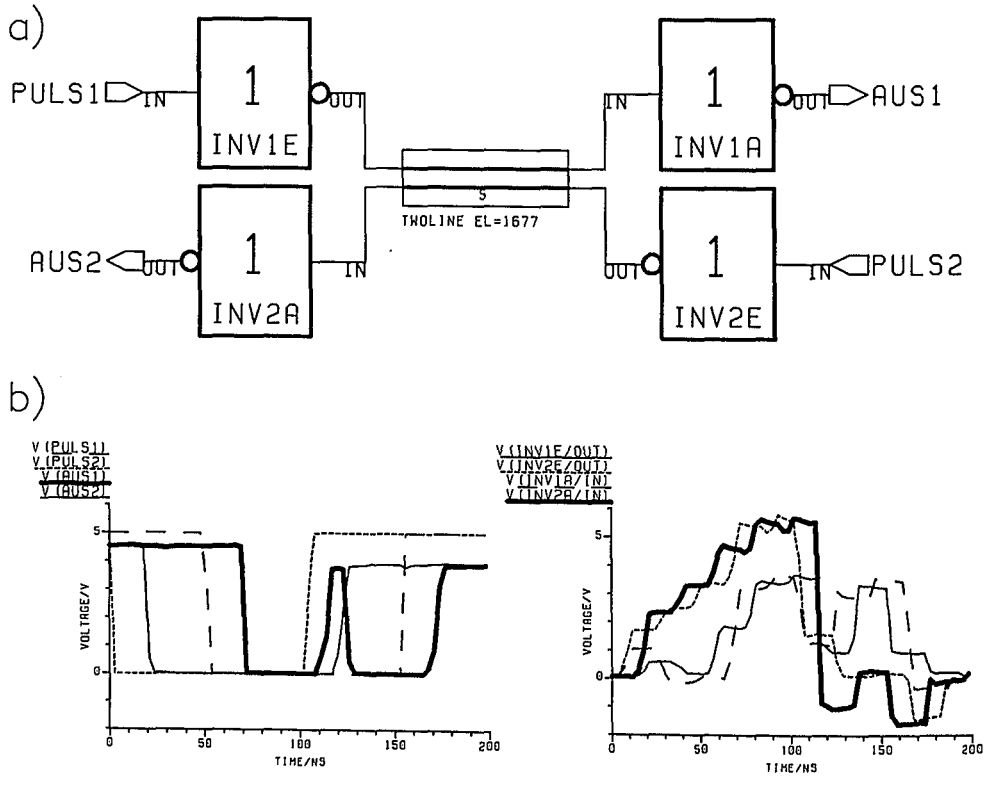


Bild 4.8: Beispiel einer Circuit–Simulation a) Schaltung b) Simulationsergebnis

Vor der Circuit–Simulation müssen folgende Grundeinstellungen vorgenommen werden:

- SETUP: Festlegung von Vorbesetzungen u.a. der Darstellungsform der Ergebnisse (welche Signale wie darstellen). In der Regel werden alle Knotenpotentiale und Zweigströme in einen Speicher geschrieben; es lassen sich somit hinterher (z. B. mit CHART–Mentor–Graphics) alle Signalspannungen und Zweigströme darstellen.

- FORCE: Festlegung der Eingangssignale (u. a. Signalform).

- READ MODELS: Festlegung der Modelle durch Laden der Modellbibliothek mit den entsprechenden Modellparametern der Halbleiterelemente und Makromodelle.

- ANALYSIS: Festlegung der Art der Analyse (Frequenzbereich, Zeitbereich, Rauschanalyse, u. a.).

Bei der Schaltungssimulation können Kennlinienverläufe und Temperaturgänge je nach Schaltungskomplexität relativ rasch ermittelt werden. Es lassen sich empfindliche Einflußparameter einer Schaltung leicht separieren und deren Auswirkungen aufdecken. In einem Circuit–Simulator sind virtuell alle üblichen Labormeßgeräte wie z.B. DC/AC–Voltmeter, DC/AC–Amperemeter, Kennlinienschreiber, Oszilloskope, Spektrumanalysator, Netzwerkanalysator enthalten. Jede beliebige Stelle einer Schaltung ist für jedes "Meßgerät" leicht zugänglich.

Zur Logik–Simulation: Ein Logiksimulator analysiert eine Logikschaltung am Modell, so wie der Logikanalysator als Meßgerät an der realen Hardware eine Logikanalyse durchführt. Bei gegebenen Eingangsstimuli werden die zeitabhängigen Logikzustände an allen Netzen in der Schaltung und an deren Ausgängen ermittelt. Die Darstellung der Logikzustände erfolgt in einem Zeitdiagramm (Trace–Darstellung) oder in einer Listendarstellung.

Heutiger Stand der Technik bei Logiksimulatoren ist die Ereignissteuerung und die Beschränkung auf "aktive" Teilnetzwerke. Wird ein Netz von einer Zustandsänderung, einem Ereignis erfaßt, so wird dies in eine sogenannte "Event–Queue" eingetragen. Wirkt das Netz auf einen Eingang eines Schaltungselementes, so werden über deren Fan–Out–Liste die vom Ereignis erfaßten nachfolgenden Netze ermittelt. Der Steuerteil des Logiksimulators arbeitet die "Event–Queue" solange ab, bis die Schaltung "ruhig" ist. Es wird dann sofort die nächste Ereignisänderung bearbeitet, die durch die Eingangssignalbelegung vorgegeben ist. In Bild 4.9 ist die Fortpflanzung eines "Ereignisses" veranschaulicht.

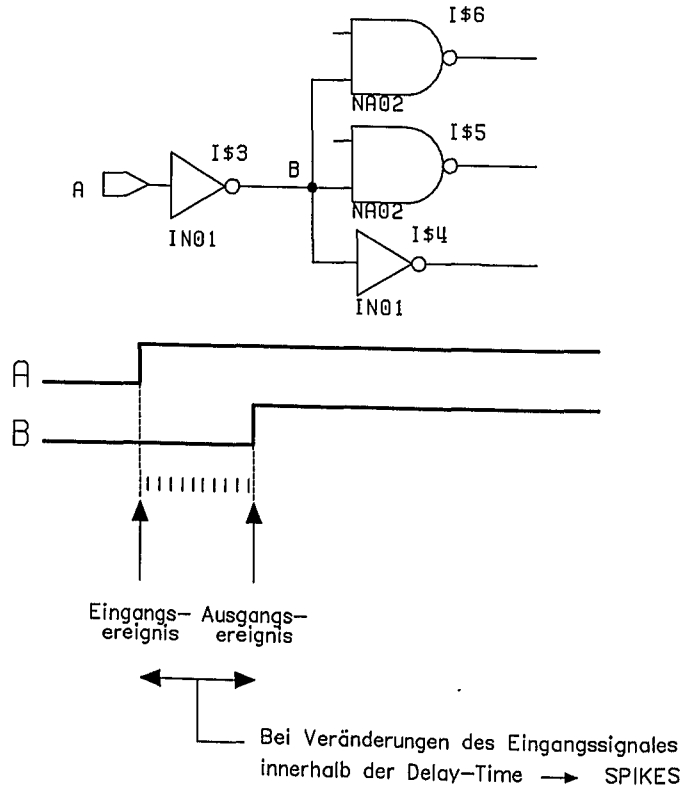


Bild 4.9: Zur Fortpflanzung eines "Ereignisses" und zur Entstehung von Spikes

Bei mehreren Ausgängen, die auf ein Netz wirken, ist die Treiberstärke der Ausgangspins (DRIVE-Property: S-Strong, R-Resistive, Z-High-Impedance, I-Indeterminate) an den jeweiligen Ausgangsnetzen maßgebend. Ein TTL-LOW-Ausgang ist beispielsweise niederohmig und damit "Strong", ein Open-Collector-HIGH-Ausgang ist "Resistive", ein abgeschaltetes Transfergate ist "Indeterminate". Bild 4.10 zeigt den resultierenden Logikzustand bei zwei auf dasselbe Netz einwirkenden Signalen mit unterschiedlicher Treiberstärke.

Simulationszustände

Drive Strength	LOW (0)	HIGH (1)	UNKNOWN (X)
Strong (S)	0S	1S	XS
Resistive (R)	0R	1R	XR
High Impedance (Z)	0Z	1Z	XZ
Indeterminate (I)	0I	1I	XI

0Z	0Z											
XZ	XZ	XZ										
1Z	XZ	XZ	1Z									
0R	0R	0R	0R	0R								
XR	XR	XR	XR	XR	XR							
1R	1R	1R	1R	XR	XR	1R						
0I	0I	XI	XI	0I	XI	XI	0I					
XI	XI	XI	XI	XI	XI	XI	XI	XI				
1I	XI	XI	1I	XI	XI	1I	XI	XI	1I			
0S	0S	0S	0S	0S	0S	0S	0S	XS	XS	0S		
XS	XS	XS	XS	XS	XS	XS	XS	XS	XS	XS	XS	
1S	1S	1S	1S	1S	1S	1S	XS	XS	1S	XS	XS	1S
	0Z	XZ	1Z	0R	XR	1R	0I	XI	1I	0S	XS	1S

Bild 4.10: Logikzustand eines Netzes in Abhängigkeit der Treiberstärken

Ähnlich wie bei der Circuit-Simulation können auch bei der Logiksimulation verschiedene Modelle verwendet werden. Für Funktionsmodelle sind auch Makromodelle möglich, deren Ein-/Ausgangsverhalten in kompakter Form als Funktionstabelle (mit Pin-to-Pin Timing) oder softwaremäßig als BLM (Behavioral Language Model) definiert wird. BLM sind durch Softwareprozeduren (Pascal oder C) beschriebene Funktionsblöcke. Mit derartigen Makromodellen lassen sich komplexe Funktionsblöcke z.B. auch Mikroprozessorbausteine und Peripheriebausteine effizient beschreiben. Mit großer Sorgfalt muß bei der Modellbildung auf die geeignete Nachbildung des Timing-Verhaltens geachtet werden.

Mit besonderen Meßeinrichtungen können reale Hardwarebausteine direkt in die Simulation einbezogen werden. Eine derartige Hardware-Modelling-Box liefert die Ausgangssignale eines Bausteins für eine gegebene Eingangssignalbelegung. Damit lassen sich ohne große Vorarbeiten auch komplexe Bausteine bei der Simulation berücksichtigen.

Bild 4.11 zeigt den prinzipiellen Ablauf der Logiksimulation.

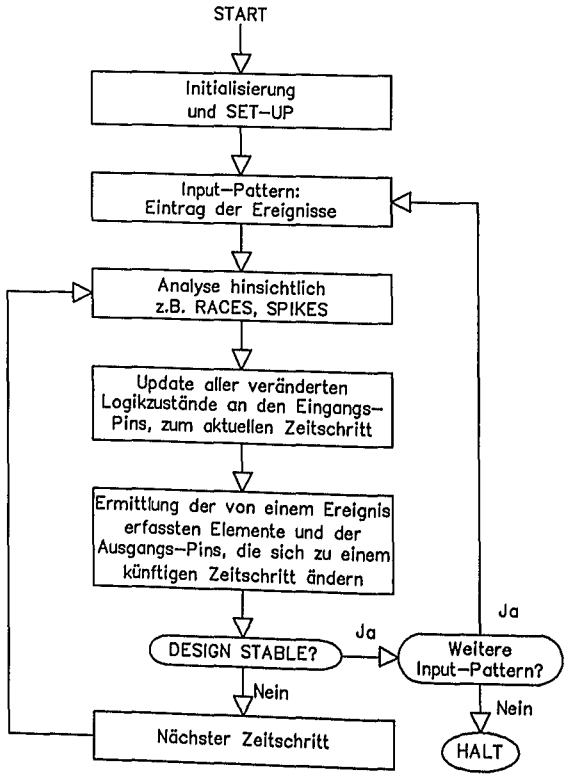


Bild 4.11: Zum Ablauf der Logiksimulation

Mögliche Fehlerzustände werden bei der Logiksimulation gesondert vermerkt. Folgende typische Fehler können dabei auftreten:

- Spikes — treten z.B. dann an Ausgängen von Schaltungen auf, wenn deren Eingangsbedingungen sich ändern, noch während die Delay–Zeit des Schaltungselements andauert (Bild 4.9).

- Races — treten z.B. dann in asynchroner Logik auf, wenn zwei oder mehr Rückführungssignale sich "gleichzeitig" ändern. Das "gleichzeitig" hängt von der Signallaufzeit an den jeweiligen Netzen ab. Es gibt kritische und unkritische Races. Bei unkritischen Races wirken sich unterschiedliche Reihenfolgen des Eintreffens des Rückwirkungsereignisses nicht aus.

- Hazards — treten z.B. dann auf, wenn an einem AND– oder NAND–Gate "gleichzeitig" sich mehr als zwei Eingangszustände in entgegengesetzter Richtung ändern (Bild 4.12).

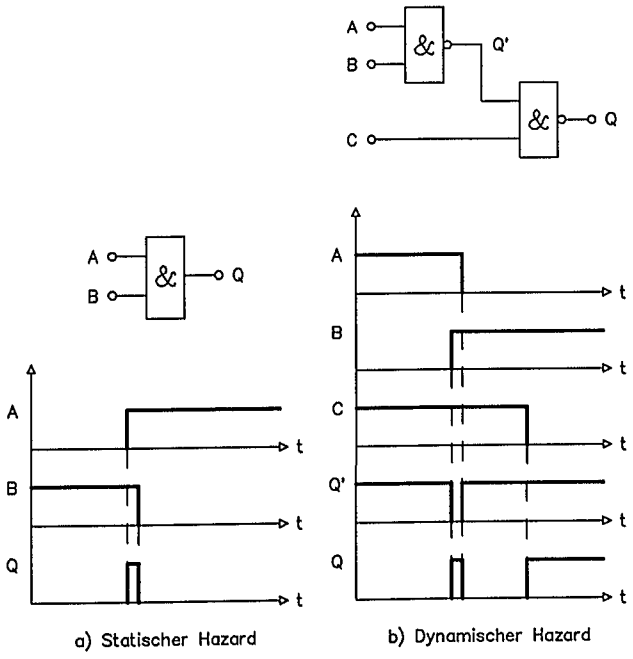


Bild 4.12: Zur Entstehung eines Hazards

Vor Ausführung der Logiksimulation müssen ebenfalls wie bei der Circuit–Simulation Vorbesetzungen vorgenommen werden:

- SETUP — Festlegung der Darstellungsform der Ergebnisse. Möglich ist eine Listen– oder Tracedarstellung.
- FORCE — Festlegen der Eingangssignale.

Die Eingangssignale werden am zweckmäßigsten mit einem Text–Editor erstellt und in DO–Files abgelegt. In der Regel gibt es hierfür eine eigene Sprache zur Festlegung der Eingangsstimuli (z.B. Interactive Stimulus Language). Damit lassen sich auch bequem komplexe Signalsequenzen relativ einfach formulieren. Auf diese Weise in einem File abgelegte Input–Pattern können bei künftig erforderlichen Modifikationen entsprechend editiert werden. Bei komplexen Digitalschaltungen ist es sehr schwierig, Eingangsstimuli so festzulegen, daß möglichst auch alle denkbaren Fehler aufgedeckt werden. Zur Überwindung dieses Problems gibt es eigene Werkzeuge, die den Anwender unterstützen, die Effizienz seiner Eingangsstimuli zu überprüfen. Darauf wird in einem späteren Abschnitt noch eingegangen.

Bild 4.13 zeigt ein Beispiel der Logiksimulation einer einfachen Digitalschaltung. Um auf den Anfangszustand wieder zurückgreifen zu können, muß vor dem ersten RUN der Anfangszustand mit "SAVE STATE FileName" abgespeichert werden. Später kann der Anfangszustand mit "RESTORE STATE FileName" wieder geladen und die Simulation erneut von Zeitschritt 0 gestartet werden, wenn z.B. zusätzliche Signale dargestellt werden sollen. Möglich ist auch das Abspeichern der laufenden Signalzustände in einer gewissen Speichertiefe (Simulation History). Damit können beliebige Signale am Ende zurückverfolgt werden (Backtracing). Auch ist das Abspeichern des Logikzustands einer Schaltung zu einem beliebigen Zeitpunkt möglich. Beim Laden dieses Signalzustands kann darauf wieder aufgesetzt und die Simulation fortgesetzt werden.

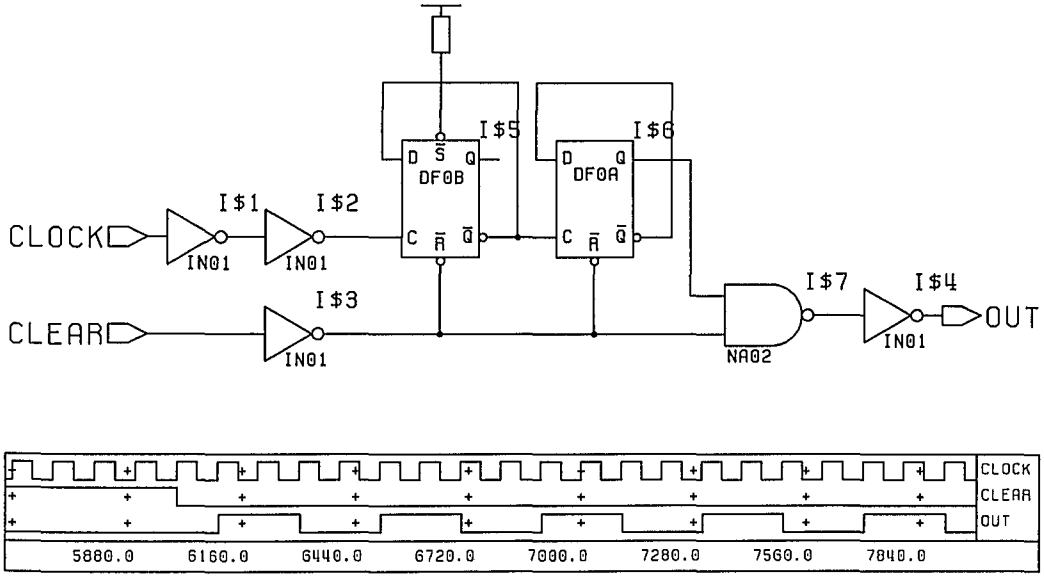


Bild 4.13: Beispiel der Logiksimulation einer einfachen Digitalschaltung

Heutige Logiksimulatoren sind in der Lage sehr komplexe Schaltungen mit bis zu einigen 100.000 Gatterfunktionen zu simulieren. Die Simulationsgeschwindigkeit beträgt ca. 2000 EPS/MIPS (Events per Sec/Million Instructions per Sec). Mit Simulationsbeschleunigern lassen sich bis zu ca. 100.000 EPS/MIPS erreichen. Simulationsbeschleuniger sind eigene Spezialrechner, in denen der Logiksimulator hardwaremäßig realisiert ist. Die Simulationszeit hängt ab von der Anzahl der Ereignisse, die in der Schaltung zu verarbeiten sind. Allerdings können Simulationsbeschleuniger nur auf der Ebene von Schaltungsprimitiven simulieren.

Die Simulationsgeschwindigkeit wird ganz wesentlich auch von den Modellen beeinflußt. Bild 4.14 zeigt den optimalen Einsatzbereich der verschiedenen Modellierungsmöglichkeiten in Abhängigkeit von der Darstellungsebene.

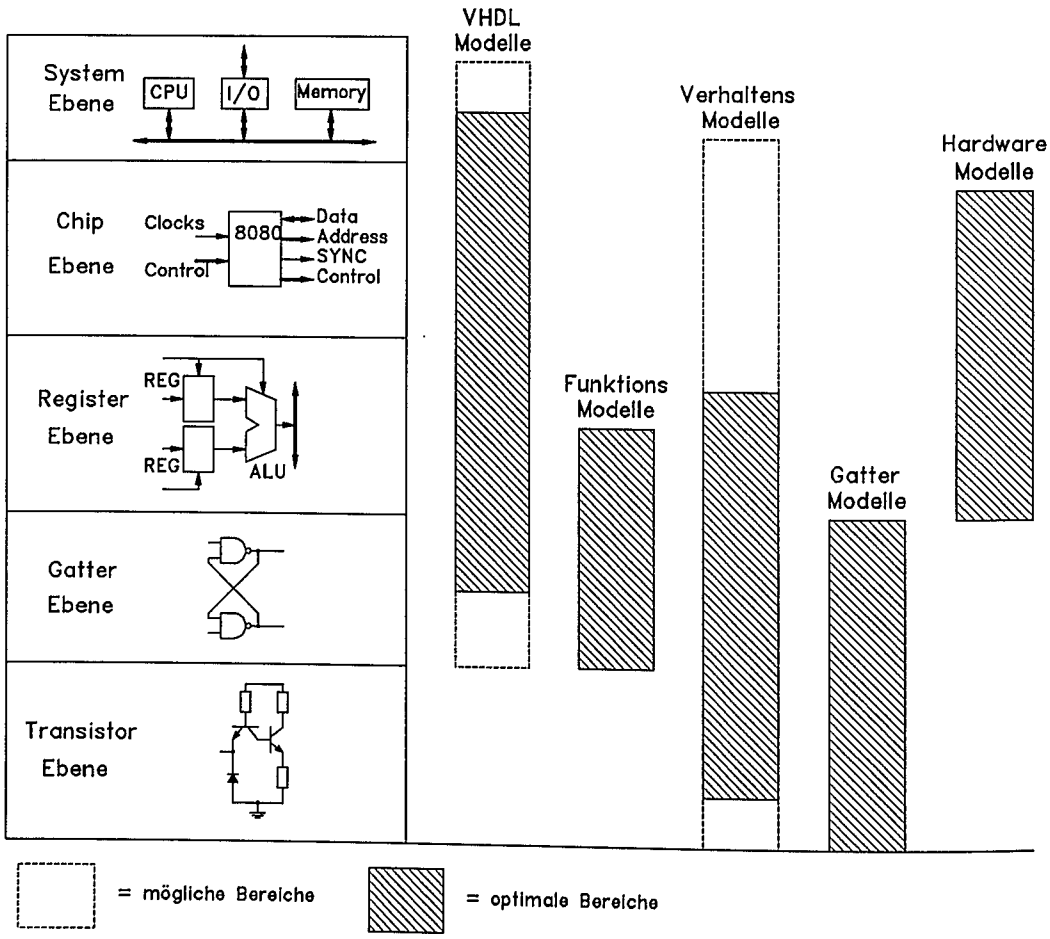


Bild 4.14: Zur Auswahl der Modelle in Abhängigkeit der Darstellungsebenen eines Entwurfs

4.3 Entwurfswerkzeuge für programmierbare Logikbausteine

Grundsätzlich können digitale Logikschaltungen in einer UND/ODER-Struktur realisiert werden. Bei programmierbaren Logikschaltungen - den PLD - werden Eingangssignale nach Durchlaufen von Eingangstreibern in einer UND/ODER- Matrixstruktur zur Realisierung komplexer Boolescher Gleichungen miteinander verknüpft (Bild 4.15). Bei PLA ist sowohl die UND-, als auch die ODER- Struktur programmierbar; bei PAL nur die UND-Struktur. Die ODER-Struktur ist dabei fest konfiguriert. Die kundenspezifische Verknüpfung erfolgt durch "Fusable Links" oder durch "Floating Gates". Bei "Fusable Links" wird die UND-Matrix durch Zerstören ("Brennen") von Polysilizium-Brücken programmiert. Der Vorgang ist nicht reversibel. "Floating Gates" werden mit MOSFET realisiert. Die Programmierung erfolgt durch Speicherung entsprechender Ladungen. Dieses Verfahren hat sich bereits bei EPROM bewährt. Durch Bestrahlung mit UV-Licht läßt sich der ursprüngliche Zustand wieder herstellen.

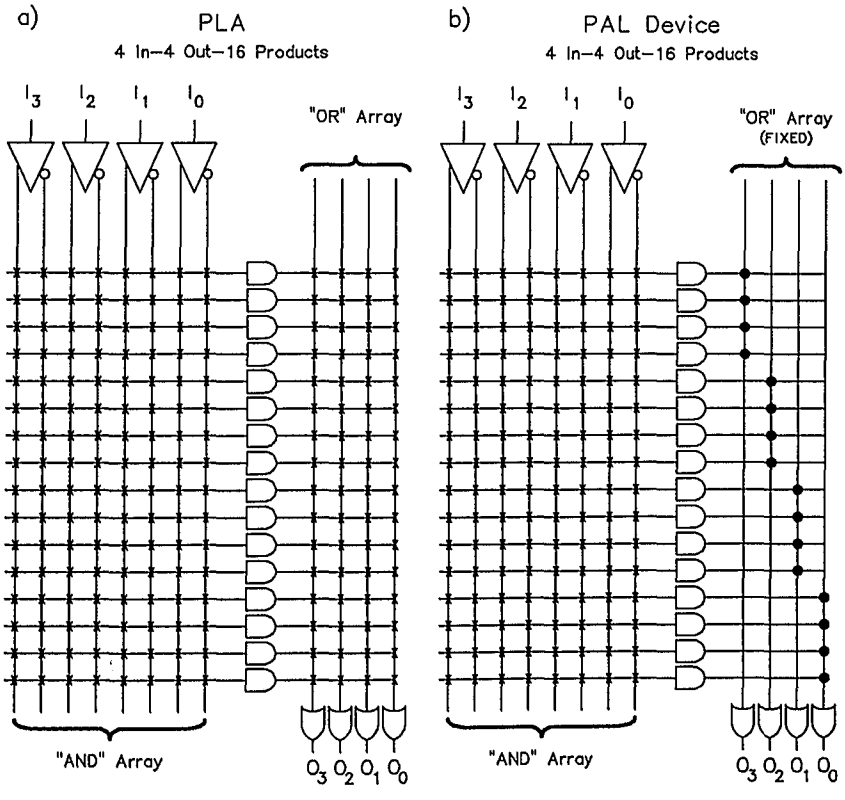


Bild 4.15: Grundsätzlicher Aufbau eines PLA- bzw.PAL-Bausteins

Bild 4.16 veranschaulicht die beiden Programmierarten "Fusable Links" und "Floating Gates". "Fusable Links" werden durch physikalisch irreversibles Zerstören von Polysilizium–Brücken programmiert. Bei "Floating Gates" erfolgt die Programmierung durch reversible Ladungsspeicherung in MOSFET.

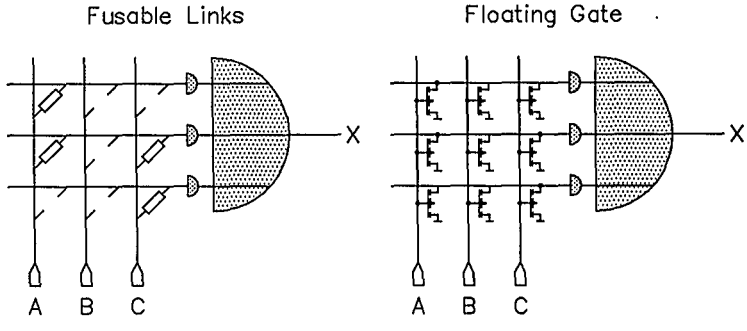


Bild 4.16: Zur Programmierung von PLD

Durch konfigurierbare I/O–Blöcke mit Registerfunktionen kann die Flexibilität des Einsatzes der PLD wesentlich erhöht werden. Damit sind auch Zustandsautomaten, wie sie in Abschnitt 2.4 beschrieben sind, realisierbar. Bild 4.17 zeigt einen Ausschnitt aus einer Matrixstruktur erweitert um ein Registerelement. Mit derartigen Registerblöcken lassen sich auch Schaltwerke und Zustandsautomaten verwirklichen.

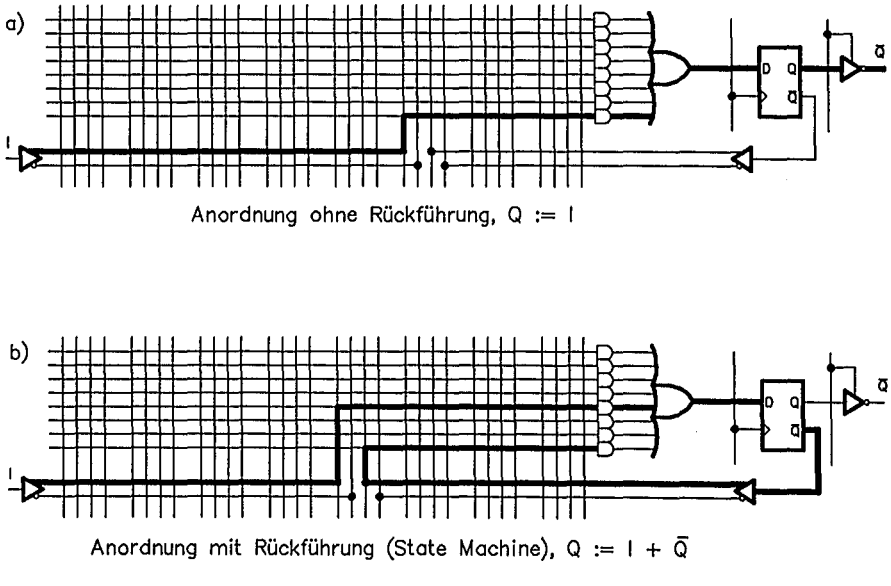


Bild 4.17: Erweiterung von PLD um Registerfunktionen

In Bild 4.18 ist ein Zustandsautomat, realisiert mit einem PLD, prinzipiell dargestellt.

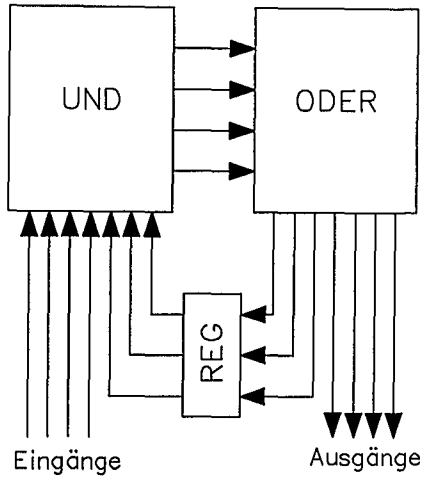


Bild 4.18: Zustandsautomat realisiert mit einem PLD; die UND/ODER-Matrizen sind programmierbar

Allgemein läßt sich die sogenannte "Glue Logic" sehr kompakt mit PLD realisieren. Bild 4.19 zeigt dies anhand eines Beispiels. Aufgabe und Zielsetzung eines PLD-Synthesewerkzeugs ist in Bild 4.20 dargestellt. Ausgehend vom Schaltplan oder von der Beschreibung eines Zustandsautomaten oder einer gegebenen Funktionstabelle oder gegebenen Booleschen Gleichungen wird automatisch ein für das spezielle Problem geeignetes PLD ausgewählt. Weiterhin werden die JEDEC-Daten erzeugt, mit denen schließlich der Baustein mit einem speziellen Programmiergerät programmiert werden kann. Mit einem PLD-Synthesewerkzeug lassen sich auch Zustandsautomaten nach dem Schema von Bild 4.18 entwerfen. Tabelle 4.1 zeigt das Eingabeformat zur Festlegung der Eingangssignale, der Ausgangssignale, der Zustandsübergangstabelle und der Zustandscodierung für das Entwurfswerkzeug LOG/iC an dem konkreten Beispiel der Ablaufsteuerung zur Erzeugung taktsynchroner Steuerimpulse in Bild 2.21. Das Logiksyntheseprogramm liefert die Booleschen Gleichungen für die Eingänge des Zustandsregisters und die Booleschen Gleichungen für die Ausgangssignale. Weiterhin kann aus einer Vorschlagliste ein geeigneter PLD-Baustein zur Realisierung des Zustandsautomaten ausgewählt werden. Die Ergebnisse der Logiksynthese sind in Tabelle 4.2 dargestellt.

LOG/iC: Produkt der Firma ISDATA, Karlsruhe

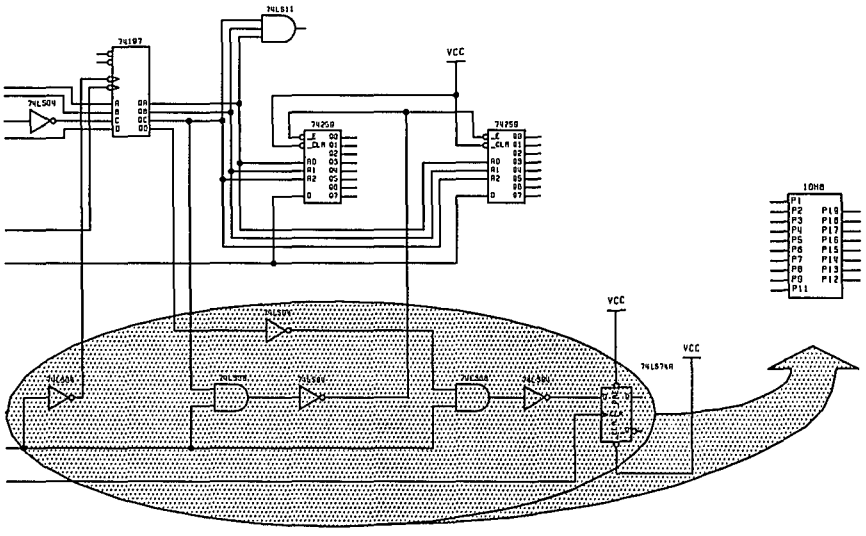


Bild 4.19: "Glue Logic" wird zu einem PLD zusammengefaßt

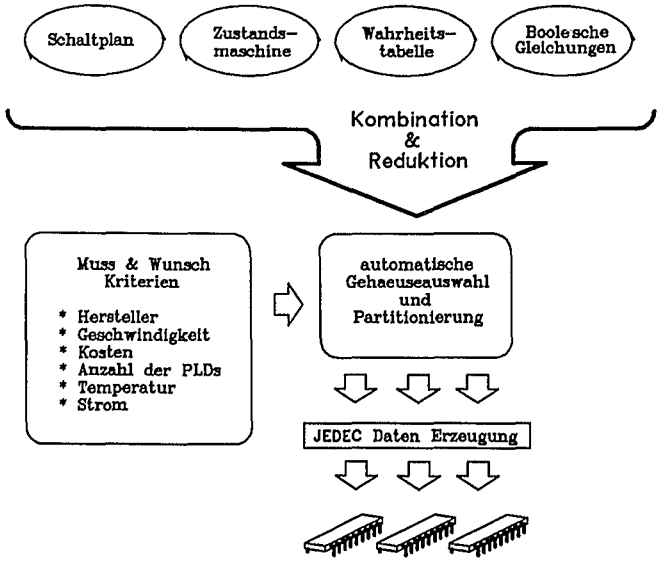


Bild 4.20: Übersicht über die Möglichkeiten eines PLD–Synthese–Werkzeugs

Tabelle 4.1: Eingabeformat für das Beispiel in Bild 2.22 für LOG/iC

```
*X-NAMES                                    EINGANGSSIGNALE
START,X1,X2,Q3,Q4,Q5;
;
*Y-NAMES                                    AUSGANGSSIGNALE
R,Y;
;
*FLOW-TABLE
RELEVANT = START,X1,X2,Q3,Q4,Q5;    RELEVANTE EINGANGSSIGNALE
;
;STATE 1 , RUHEZUSTAND
S1 , X 0 - - - - - , Y 0 0 , F1; RUHEZUSTAND
S1 , X 1 - - - - - , Y 1 0 , F2; START
;
;STATE 2 , VORBEREITUNGSZUSTAND
S2 , X - 0 0 - - - , Y 0 0 , F1;  0 us
S2 , X REST        , Y 0 1 , F3;
;
;STATE 3 , AUSFUEHRUNGSZUSTAND
S3 , X - 1 0 0 1 0 , Y 0 0 , F1; 16 us
S3 , X - 0 1 1 1 0 , Y 0 0 , F1; 24 us
S3 , X - 1 1 0 1 1 , Y 0 0 , F1; 48 us
S3 , X REST        , Y 0 1 , F3;
;
*STATE-ASSIGNMENT
BITS = 2 ;
S1 = 00 ;       ZUSTANDSCODIERUNG FESTLEGEN
S2 = 01 ;
S3 = 1- ;
;
*FLIPFLOPS
D-FLIPFLOPS
;
*PAL
TYPE = HYPERPAL;
;
*RUN-CONTROL
LISTING=PLOT,EQUATIONS,FUSE-PLOT;
;
*END
```

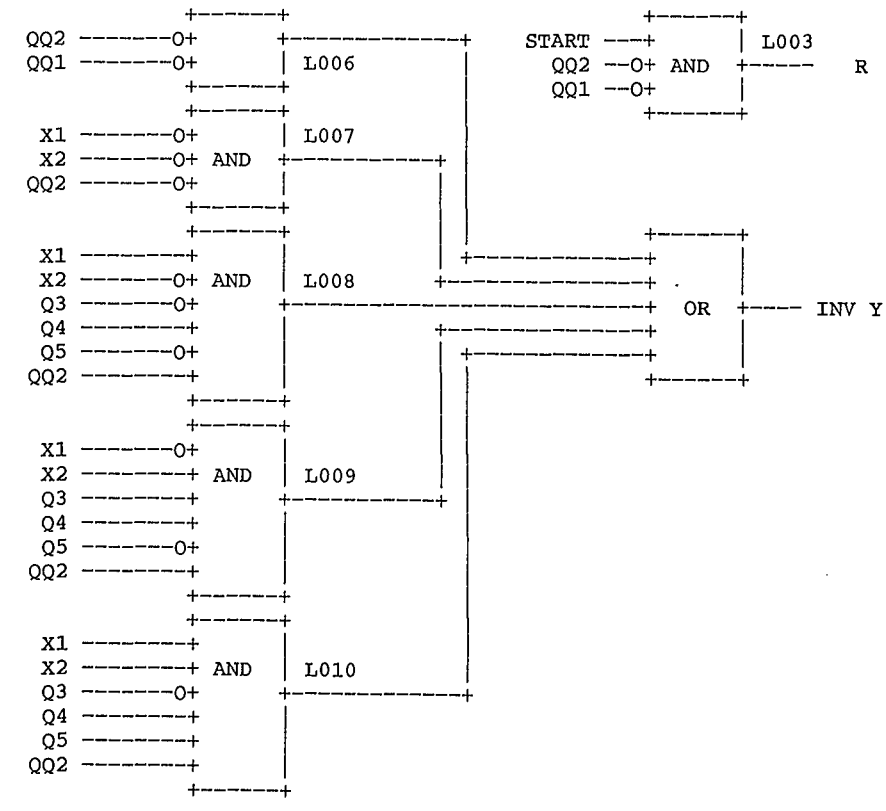
Tabelle 4.2: Ergebnisse der Logiksynthese des Beispiels in Bild 2.22

```
        IDENTICAL FUNCTIONS:
        --------------------
        QQ2.D =   Y
        QQ1.D =   R

   ************************************************
   ***              BOOLEAN EQUATIONS             ***
   ************************************************

 R       =  START  & /QQ2    & /QQ1    ;


/Y       = /QQ2    & /QQ1
         + /X1     & /X2     & /QQ2
         +  X1     & /X2     & /Q3     &  Q4     & /Q5     &  QQ2
         + /X1     &  X2     &  Q3     &  Q4     & /Q5     &  QQ2
         +  X1     &  X2     & /Q3     &  Q4     &  Q5     &  QQ2    ;
```



Im Jahre 1985 kamen die elektrisch löschbaren EEPLD (GAL) auf den Markt. Beim heutigen Entwicklungsstand können GAL etwa 100mal gelöscht und neu programmiert werden. Ebenso neu auf den Markt sind die Logic-Cell-Arrays (LCA). Diese bestehen aus konfigurierbaren Logikzellen und einer Verdrahtungsmatrix. Die Verdrahtungswege sind durch spezielle Transistorelemente "routbar". Je nach dem Schaltungsstand der Transistorelemente ergeben sich verschiedene Verdrahtungswege. Dieser Aufbau geht schon mehr in Richtung Gate-Array (Bild 4.22). Der Vorteil der LCA liegt in der Incircuit-Umkonfigurierbarkeit des Bausteins, während alle anderen PLD für die Programmierung in ein eigenes Programmiergerät gesteckt werden müssen.

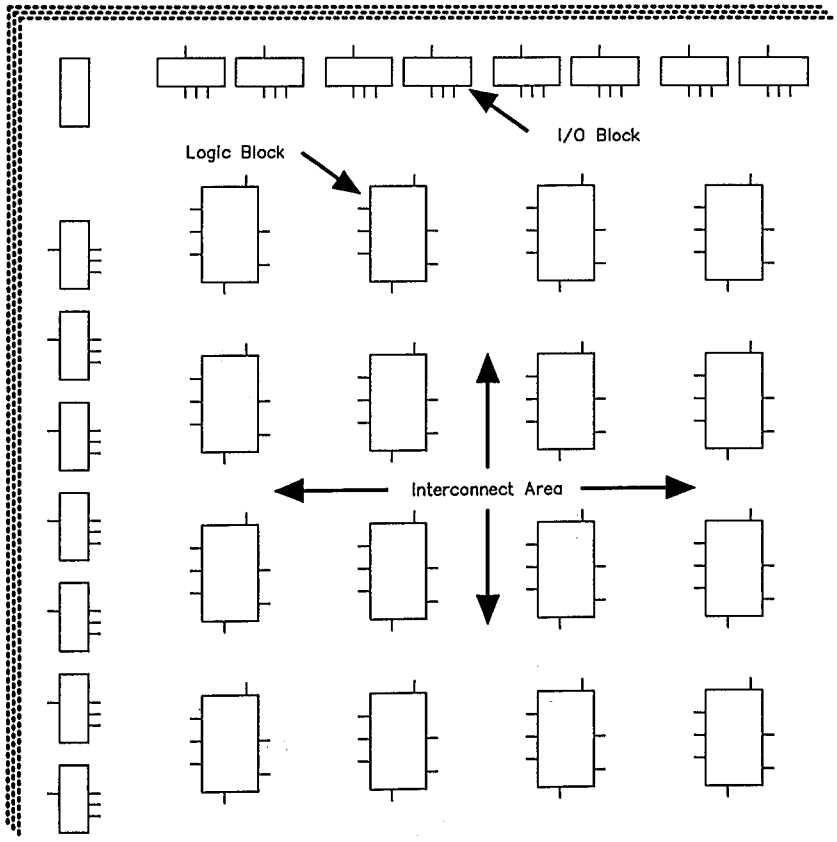


Bild 4.21: Grundsätzlicher Aufbau von LCA-Bausteinen

In Bild 4.22 ist der konfigurierbare Logik–Block skizziert, ebenso die Möglichkeit des Programmierens von Verbindungen. Der konfigurierbare Logikblock besteht aus einem Registerelement und einem kombinatorischen Logikblock. Die Konfigurierung erfolgt über programmierbare Multiplexer. Damit lassen sich Signalpfade einstellen.

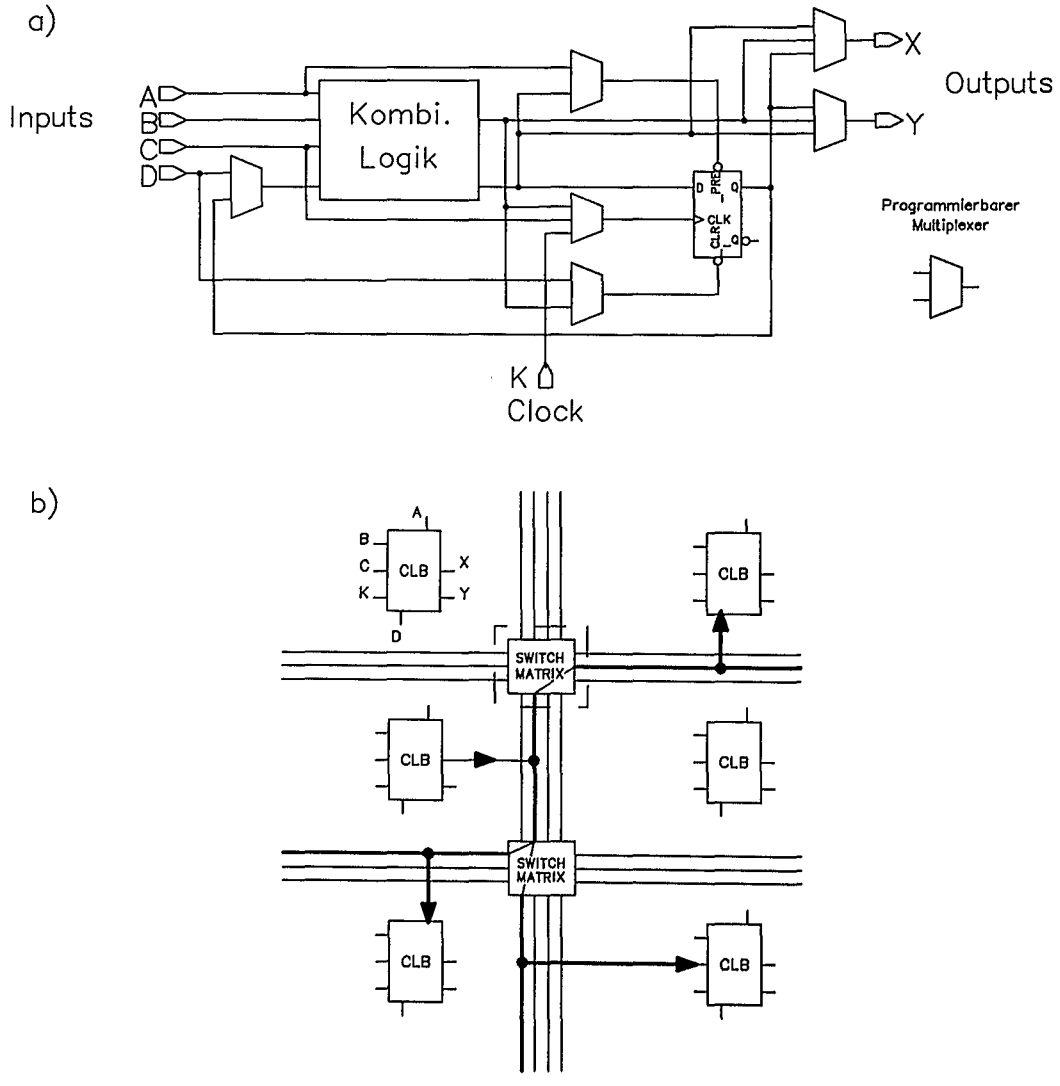


Bild 4.22: Grundelemente eines LCA–Bausteins (Prinzipskizze)
a) Konfigurierbarer Logik–Block
b) Programmierbarkeit der Verbindungen

Tabelle 4.3 zeigt als Beispiel einen 8-Bit-Multiplexer. Die Beschreibung erfolgt in Form einer Funktionstabelle mit dem Logiksynthesewerkzeug LOG/iC. Realisiert werden soll der 8-Bit-Multiplexer mit einem LCA-Baustein. Das Logiksynthesewerkzeug erstellt u.a. die Programmierdaten für den ausgewählten LCA-Baustein.

Tabelle 4.3: Eingabeformat zur Logiksynthese eines 8-Bit-Multiplexers

Data Set: MUX8.DCB

```
*IDENTIFICATION
 8 BIT Multiplexer
 LOG/IC Example

*DECLARATIONS
 X-VARIABLES = 11;
 Y-VARIABLES = 1;

*X-NAMES
 S (2..0),   ! Multiplexer control input
 FB(3..0),   ! Lower 4 Bits
 FA(3..0);   ! Upper 4 Bits

*Y-NAMES
 DABE_N;

*RUN-CONTROL
 LISTING = GATES, EQUATIONS;
 STAGES  = 3,7;
 POST    = LCA;
 GATLIB  = /LOGIC/LCA20XX5;

*FUNCTION-TABLE
 $(S(2..0)),((FB(3..0),FA(3..0))) : DABE_N;
 0D , ---- ---0 : 0; FA0
 1D , ---- --0- : 0; FA1
 2D , ---- -0-- : 0; FA2
 3D , ---- 0--- : 0; FA3
 4D , ---0 ---- : 0; FB0
 5D , --0- ---- : 0; FB1
 6D , -0-- ---- : 0; FB2
 7D , 0--- ---- : 0; FB3
 REST           : 1;

*END
```

FPGA (Field Programmbale Gate Arrays) schlagen eine Brücke zwischen der flexiblen Programmierfähigkeit der PLD und der Leistungsfähigkeit der Masken–Gate–Arrays (Bild 1.6). Ein FPGA–Baustein benötigt für die Personalisierung keine Masken. Er besteht aus Logik–Modulreihen mit programmierbaren Verbindungen in anwendungsspezifischen Schaltkreisen. Die Module sind keine starren Strukturen, wie bei den üblichen PLD, sondern sie sind vielmehr konfigurierbar. Erst nach der Programmierung erhalten die Schaltkreise eine bestimmte Funktion: u.a. AND, OR, XOR, ... , Flipflops, Multiplexer, Addierer. Für die Festlegung einer Funktion stehen Hard– und Softmakros zur Verfügung. Zwischen den Logik–Modulreihen und –spalten sind die Verdrahtungskanäle angeordnet. Die Verbindung zwischen den Leiterbahnen (Tracks) ist programmierbar mit PLICE–Elementen (PLICE: Programmable–Low–Impedance –Element). Die Verbindungen sind in Antifuse–Technologie ("Schweißpunkt"– Technologie) ausgeführt. Ein PLICE– Element stellt einen Zweipol dar und wirkt wie ein programmierbarer Widerstand. Es wird eine "Vias"–Verbindung zwischen Leiterbahnen hergestellt. Um den Kern aus Logik–Modulreihen befindet sich ein Ring aus E/A–Modulen und Pads.

Mit FPGA lassen sich vielfältige Logikfunktionen realisieren. Die Möglichkeiten gehen deutlich sowohl hinsichtlich der Funktionalität als auch hinsichtlich der Komplexität über die Möglichkeiten von PAL hinaus. Der Entwicklungsaufwand ist gegenüber den Masken–Gate–Arrays erheblich geringer; es entfällt beispielsweise der Aufwand, um Fertigungsfehler bei Masken–Gate–Arrays aufzudecken. Vor dem Versand wird jede Funktion außer der Antifuse getestet. Eine Antifuse–Kontrolle erfolgt während des Programmier–Zyklusses. Wegen der besseren Ausnutzung der Siliziumfläche und anderer Vorteile lohnt sich der Mehraufwand in der Entwicklung von Masken–Gate–Arrays bei höherer Schaltungskomplexität und bei höheren Stückzahlen. Oftmals wird zur Realisierung eines Prototypen eine Logikfunktion zunächst mit FPGA aufgebaut. Für die Serie werden dann aus Platz– und Kostengründen Gate–Array–Bausteine oder Standard–Cell–Bausteine entwickelt.

Allgemeine Anmerkung: PLD sind vollständig vorgefertigte und vom Hersteller applikationsunabhängig getestete IC. Es werden keine Masken benötigt. Erforderlich ist ein spezielles Programmiergerät und Entwurfs–Software für die kundenspezifische Programmierung des Bausteines. Nach Eingabe des logischen Verhaltens der Ein–/Ausgangs–Pins durch Zuweisungen und Boolesche Gleichungen oder Funktionstabellen, erfolgt durch die Entwurfs–Software die optimale Umsetzung in die programmierbare Form des ausgewählten PLD.

PLD haben sich bereits einen ansehnlichen Marktanteil erobert. Die Komplexität reicht heute bis ca. einige 1000 Gatterfunktionen. Besonders vorteilhaft ist die schnelle und kostengünstige "Personalisierung" des Bausteins. Das spart Entwicklungszeit und Entwicklungskosten. In vielen Fällen werden Standard–IC (TTL–, CMOS–IC) durch PLD ersetzt.

4.4 Werkzeuge zur Fehlersimulation und Timing–Verifikation

Nach Verifikation der Schaltungsfunktion mit dem Logiksimulator muß die Schaltung im Hinblick auf kritische Zeitbedingungen untersucht werden. Eine Worst–Case–Timing–Analysis ermittelt die mögliche Zeitdauer der Unbestimmtheit des Logikzustands an den Ausgängen. Beispielsweise ergeben sich Probleme an D–Flipflops, wenn der D–Eingang noch unbestimmt ist, während sich die Taktflanke von "0" auf "1" ändert (Setup–and–Hold–Time Violation). Die Unbestimmtheit eines Ausgangs ergibt sich beispielsweise aus der minimalen bzw. maximalen RISE/FALL–Time bzw. DELAY–Time. Die Timing–Analyse überprüft den Einfluß der Bauteiltoleranzen in einer "Worst–Case"–Simulation. Man unterscheidet eine dynamische bzw. eine statische Timing–Analyse.

Dynamische Timing–Analyse: Überprüft wird das funktionale Timing–Verhalten. Im Timing–Modell eines Schaltungselements sind die min/typ/max–Verzögerungszeiten von Pin zu Pin angegeben. Der "Timing–Analyser" ermittelt daraus den sich ergebenden Unsicherheitsbereich (Slack–Time) an den Netzen. In Bild 4.23a ist dies für eine einfache kombinatorische Schaltung dargestellt. Bei sequentiellen Schaltungen (Bild 4.23b) ist die Änderung am Ausgang vom Takt abhängig. Diese Art von Timing–Analyse ist abhängig vom Testpattern und damit auch der Aufdeckungsgrad der Timing–Fehler; sie ist nicht orientiert am Signalpfad. Timing–Fehler werden angezeigt für einen bestimmten Pin zu einer bestimmten Zeit.

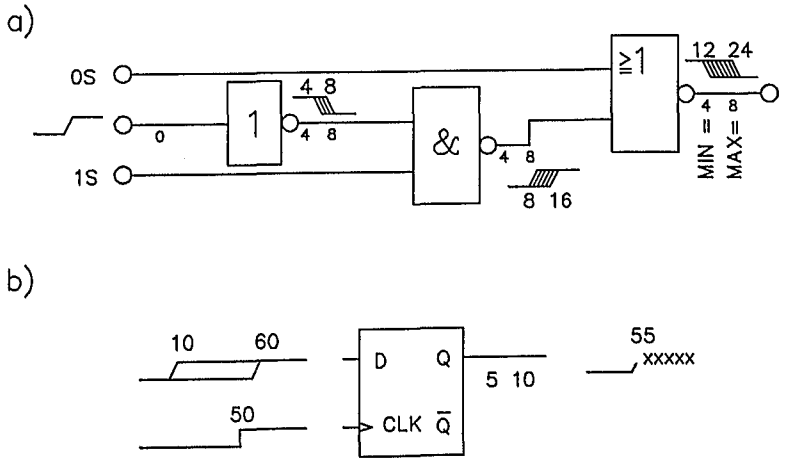


Bild 4.23: Beispiel der Timing–Analyse einer Digitalschaltung
a) kombinatorische Schaltung
b) sequentielle Schaltung

Statische Timing–Analyse: Ermittelt wird das Timing–Verhalten durch Addition der Verzögerungszeiten längs eines Signalpfads. Überprüft wird die Signallaufzeit längs eines kritischen Pfads und ob z.B. eine "Setup–and–Holdtime Violation" vorliegt. Bild 4.24 zeigt als Beispiel eine kritische–Pfad Analyse. Pfad 1 geht vom Ausgang von Register U1 über die Gatter U4/U5 zum D–Eingang von U2. Es werden alle Verzögerungszeiten längs dieses Signalpfads aufaddiert und mit dem Eintreffen des CLOCK–Ereignisses an U2 verglichen. Auch kann wiederum eine "Worst–Case"/"Best–Case"–Analyse vorgenommen werden.

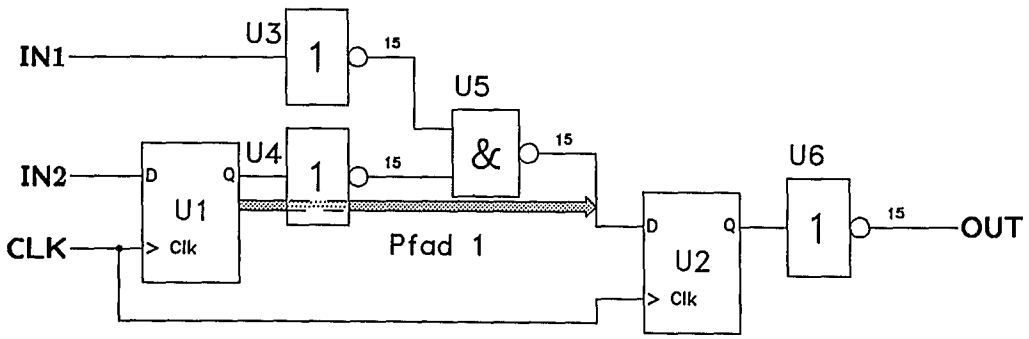
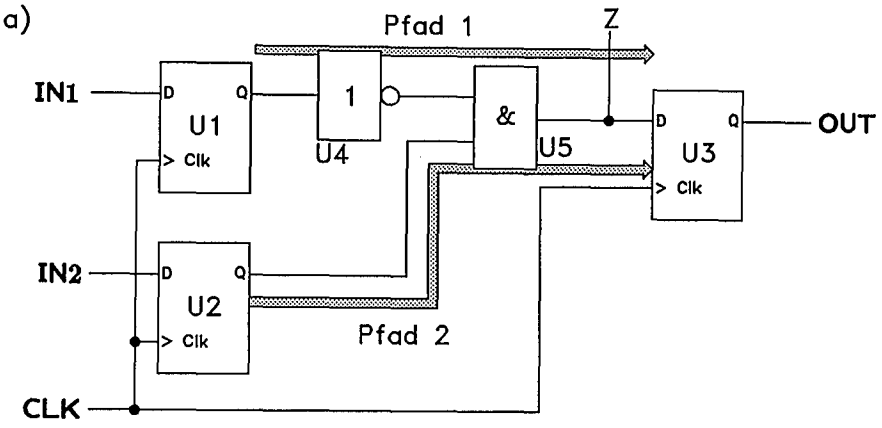


Bild 4.24: Kritische–Pfad Analyse

Die statische Timing–Analyse orientiert sich immer an einem Signalpfad, beginnend an einem Source–Flipflop und endend an einem Destination–Flipflop. Der Pfad endet immer an einem getakteten Baustein. Die statische Timing–Analyse kann keine Logiksimulation ersetzen. Damit lassen sich auch keine Races aufdecken, da deren Auftreten vom Logikverhalten abhängt. Die kritische–Pfad Analyse kann auch abhängig vom Testpattern sein, wie das Beispiel in Bild 4.25 zeigt. Je nach dem Testpattern IN1/IN2 ist Pfad1 bzw. Pfad 2 wirksam. Im Bild ist ein geeignetes Testpattern für dieses Beispiel dargestellt.

a)



b)

	IN1	IN2	OUT	U5 Ausgang (Punkt Z)
t1	0	0	0	0
t2	0	1	1	Pfad 2 => Z: 0–>1
t3	0	0	0	Pfad 2 => Z: 1–>0
t4	1	1	0	auf Spikes untersuchen
t5	0	1	1	Pfad 1 => Z: 0–>1
t6	1	1	0	Pfad 1 => Z: 1–>0

Bild 4.25: Kritische–Pfad Analyse mit zwei Signalpfaden

Im allgemeinen sind die Ergebnisse der Timing–Analyse oft zu pessimistisch, da in der Realität ein ungünstigstes Zusammenwirken der min/max–Verzögerungszeit nicht vorkommt. Dies erklärt sich schon allein aus der Tatsache, daß alle Schaltungselemente auf demselben Substrat auch denselben Einflüssen (z.B. gleicher Herstellungsprozeß, gleiche Temperatur, gleiche Versorgungsbedingungen) ausgesetzt sind.

Timing–Verifier: Der Timing–Verifier unterscheidet nur zwischen "stabilen" und "unsicheren" Zuständen an Stelle der üblichen Logikzustände. Möglich ist diese Art der Timing–Verifikation bei synchronen und sequentiellen Schaltungen. Bild 4.26 zeigt ein Beispiel. Die Darstellungsart unterscheidet sich von der des Logiksimulators. Im allgemeinen ist die Aussage eines Timing–Verifiers pessimistisch. Es werden reale Fehler, unwirksame Fehler und pessimistische Fehler aufgedeckt.

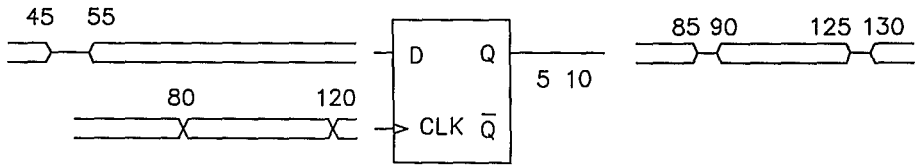


Bild 4.26: Beispiel Timing–Verifier

Zur Fehlersimulation: Ziel ist es zu ermitteln, ob mögliche Fertigungsfehler durch die gegebenen Input–Pattern aufgedeckt werden. Bei integrierten Schaltungen gibt es z.B. "Stuck–Open", und "Open–Metallization"–Fehler. Bild 4.27 zeigt beispielhaft einen physikalischen Defekt. Mit geeigneten Fehlermodellen werden Fertigungsfehler simuliert. Das einfachste Fehlermodell ist das "Stuck–at–Zero" (SA0) und "Stuck–at–One"–Fehlermodell (SA1). Ein Netz wird auf "0" oder "1" gelegt, dabei wird ermittelt, ob dieser Fehler aufgedeckt wird. Die Input–Pattern müssen so gewählt werden, daß der Aufdeckungsgrad von Fehlern möglichst hoch ist (min. 95%).

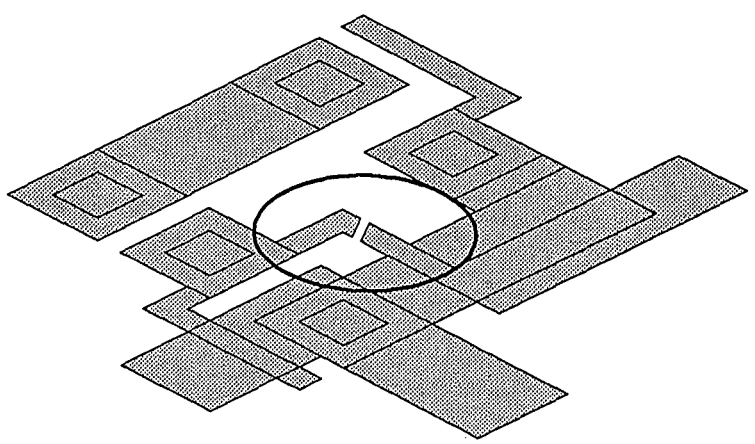


Bild 4.27: Open–Metallization–Fehler

Aufgabe des Fehlersimulators ist es, aufzuzeigen, welche Netze bei welcher Fehlerart vom vorgegebenen Input–Pattern nicht aufgedeckt werden. Den in Bild 4.27 dargestellten Fertigungsfehler gilt es beobachtbar zu machen. Unerkannte SA1– bzw. SA0–Fehler werden z.B. im Schaltplan am jeweiligen Netz gekennzeichnet.

Der Fehlersimulator arbeitet im Prinzip immer in zwei Schritten. Zunächst wird die Schaltung ohne Fehler mit den Input–Pattern an den primären Eingängen simuliert. Die Ereignisse an den primären Ausgängen werden abgespeichert. Im zweiten Schritt werden Fehler an einem beliebigen Netz in die Schaltung eingebaut. Die Schaltung wird erneut simuliert. Durch Vergleich der Ereignisse an den primären Ausgängen mit den Ergebnissen des ersten Schrittes kann festgestellt werden, ob der Fehler detektierbar ist.

Fehlersimulatoren lassen sich nach deterministischen und statistischen Methoden unterscheiden. Deterministische Fehlersimulatoren liefern exakte Ergebnisse, benötigen aber lange Simulationszeiten. Statistische Methoden sind hingegen erheblich schneller. Deterministische Fehlersimulatoren unterscheidet man hinsichtlich einer seriellen oder parallelen Vorgehensweise. Bei der seriellen Vorgehensweise wird jeweils immer nur ein einziger Fehler pro Simulationslauf in die Schaltung eingebaut. Parallele Fehlersimulatoren nutzen die volle Wortbreite eines Computers und simulieren mehrere Fehler pro Simulationslauf; sie sind aber nur auf Gatterebene einsetzbar. Statistische Fehlersimulatoren sind erheblich schneller, sie werden vor allem zum Austesten von Test–Pattern eingesetzt. Aus der Gesamtheit aller möglichen Fehler wird nur eine Untermenge simuliert und daraus die Fehlerabdeckungsrate hochgerechnet.

Die Fehlersimulation ist außerordentlich rechenintensiv. Durch geeignete Parallelverarbeitung läßt sich eine wesentliche Beschleunigung erzielen. Die Fehleranalyse bringt folgende Ergebnisse:

* Gesamtzahl der Fehler; nichttestbare Fehler (nicht testbar aufgrund der Schaltungstopologie); detektierbare Fehler; mögliche Fehler (Fehler, die durch eine Änderung von "0" oder "1" auf "Unbestimmt" entstehen).

* Nicht beobachtbare Fehler; nicht kontrollierbare Fehler.

Für die Teststimuli–Entwicklung interessant ist auch das sogenannte "Toggle–Test"–Ergebnis. Es wird angegeben welche Netze im Verlauf einer Simulation ihren logischen Zustand nicht ändern.

Zur Vorgehensweise bei der Test–Pattern–Entwicklung: Nach der funktionellen Verifikation einer Schaltung mit dem Logiksimulator erfolgt die Timing–Analyse. Anschließend müssen die Test–Stimuli für den späteren Hardware–Test entwickelt und ausgetestet werden. Aus zeitökonomischen Gründen wird zunächst oft ein statistischer Fehlersimulator eingesetzt. Zuletzt erfolgt die Ermittlung der exakten Fehlerabdeckungsrate mit einem deterministischen Fehlersimulator. Schließlich werden die notwendigen Daten für die Testerschnittstelle erzeugt. Die praktische Vorgehensweise ist in Bild 4.28 dargestellt.

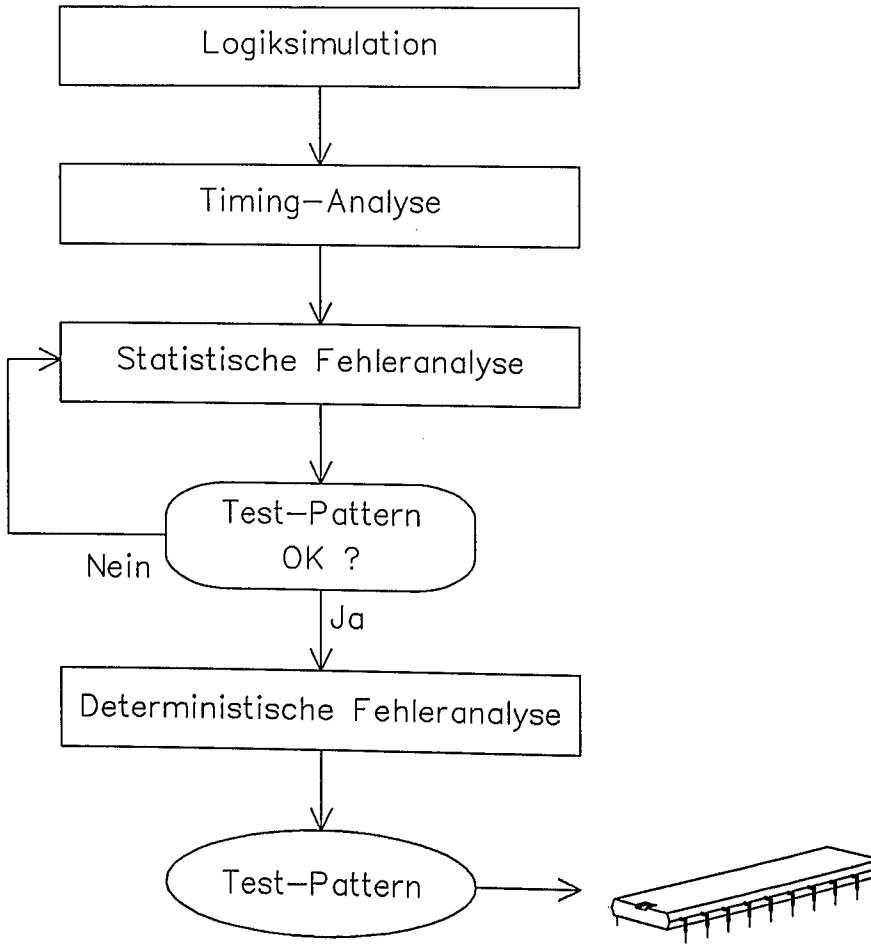


Bild 4.28: Zum Ablauf der Entwurfsverifikation

Literatur zu Kapitel 4:

/1/ EDIF Steering Committee: EDIF Electronic Design Interchange Format: Version 2 2 0, 1987

/2/ Mentor–Graphics: An Introduction to Digital Simulation. 1989

/3/ W.Voldan: Schaltungserstellung. Elektronik ASIC–Sonderheft 1988

/4/ A. Auer: Programmierbare Logikschaltungen. Heidelberg: Hüthig 1989

/5/ A. Auer: PLD–Handbuch, Tabellen und Daten. Heidelberg: Hüthig 1990

Fragen zum 4.Kapitel

F4.1 Skizzieren Sie den Designablauf beim Entwurf Integrierter Schaltungen!

F4.2 Welche drei Grundelemente enthält die symbolische Schaltungsstruktur?

F4.3 Was sind Properties und welche Bedeutung haben sie?

F4.4 Wo können Properties im Schaltplan angefügt werden und was ist dabei zu beachten?

F4.5 Nennen Sie wichtige Grundfunktionen bei der Erstellung eines Schaltplans!

F4.6 Auf welche Weise werden neue Symbole erstellt? Was muß ein Symbol alles enthalten?

F4.7 Welche Arten von Schaltungssimulatoren gibt es und in welchen Bereichen werden sie eingesetzt?

F4.8 Welche Modellierungsarten für Schaltungselemente gibt es?

F4.9 Welche Aufgabe und welchen Einsatzbereich hat ein Circuit–Simulator?

F4.10 Welche Festlegungen müssen vor der Circuit–Simulation getroffen werden?

F4.11 Wie arbeitet ein Logiksimulator?

F4.12 Welche Logikzustände eines Netzes sind in Abhängigkeit der Treiberstärke möglich?

F4.13 Welche Festlegungen müssen vor der Logiksimulation getroffen werden?

F4.14 Skizzieren Sie den Aufbau von PLA; Wodurch unterscheiden sich PLA von PAL?

F4.15 Skizzieren Sie die Realisierung eines Zustandsautomaten mit PLD!

F4.16 Was sind LCA? Skizzieren Sie deren Aufbau! Aus welchen Elementen bestehen sie?

F4.17 Was ist ein Timing–Verifier und welche Art von Analyse führt er durch?

F4.18 Was macht ein Fehlersimulator und welche Art von Analyse führt er durch?

F4.19 Skizzieren Sie die Vorgehensweise für die Entwicklung von Teststimuli zum Hardwaretest eines Bausteins?

F4.20 Welche Modellarten benötigt der Circuit–Simulator?

F4.21 Auf welche Weise können Logikfunktionen modelliert werden?

F4.22 Wie arbeitet die statische Timing–Analyse und welche Ergebnisse bringt sie?

5. Simulationsmethoden für die Schaltungsverifikation

Simulationswerkzeuge sind für den Anwender "Meßgeräte". Für den richtigen und geeigneten Einsatz ist ein hinreichendes Grundverständnis von deren Arbeits- und Funktionsweise erforderlich. Zur Vertiefung dieses Kapitels sei auf /1/ bis /8/ verwiesen.

5.1 Zur Methodik der Circuit-Simulation

Grundsätzliche Vorgehensweise: Die Netzwerkanalyse von nichtlinearen Schaltkreisen im Zeitbereich erfordert die Lösung eines nichtlinearen Differentialgleichungssystems. Durch Zeitdiskretisierung und Linearisierung der nichtlinearen Schaltungselemente erhält man ein algebraisches Gleichungssystem, das iterativ gelöst wird.

Gegeben sind die nichtlinearen Netzwerkgleichungen:

$$\mathbf{f}_1(\mathbf{x};\ \delta\mathbf{x}/\delta t;\ \mathbf{y};\ t) = \mathbf{0}\ ; \qquad (5.1)$$
$$\mathbf{f}_2(\mathbf{x};\ \mathbf{y};\ t) = \mathbf{0}\ .$$

Dabei sind $\mathbf{x}$ die Knotenpotentiale, deren Ableitung ebenfalls vorkommt; $\mathbf{y}$ sind die restlichen Knotenpotentiale.

Mit der Zeitdiskretisierung

$$\delta\mathbf{x}/\delta t = (\mathbf{x}_n - \mathbf{x}_{n-1})/h_n\ ; \qquad \text{mit } t_n = t_{n-1} + h_n \qquad (5.2)$$

erhält man folgendes System nichtlinearer algebraischer Gleichungen für den Zeitpunkt t_n:

$$\bar{\mathbf{f}}_1(\mathbf{x}_n;\ \mathbf{x}_n - \mathbf{x}_{n-1};\ \mathbf{y}_n) = \mathbf{0}\ ;$$
$$\bar{\mathbf{f}}_2(\mathbf{x}_n;\ \mathbf{y}_n) = \mathbf{0}\ . \qquad (5.3)$$

Mit $\mathbf{g} = (\bar{\mathbf{f}}_1;\ \bar{\mathbf{f}}_2)$ und $\mathbf{z}_n = (\mathbf{x}_n;\ \mathbf{y}_n)$ ergibt sich nach Taylorreihenentwicklung und Anwendung der Newton-Iterations-Methode für $\mathbf{g}(\mathbf{z}_n) = \mathbf{0}$:

$$(\delta\mathbf{g}/\delta\mathbf{z}_n^{(i-1)})\mathbf{z}_n^{(i)} = -\mathbf{g}(\mathbf{z}_n^{(i-1)}) + (\delta\mathbf{g}/\delta\mathbf{z}_n^{(i-1)})\mathbf{z}_n^{(i-1)}\ ; \qquad (5.4)$$

Diese Gleichung läßt sich mit $\mathbf{A} = (\delta\mathbf{g}/\delta\mathbf{z}_n^{(i-1)})$ (Jakobi-Matrix) auf eine programmierbare Form bringen, deren Lösung iterativ ermittelt wird:

$$\mathbf{A} * \mathbf{z}_n^{(i)} = \mathbf{b}. \qquad (5.5)$$

Bei bekanntem $\mathbf{z}_n^{i-1}$ kann aus dieser Gleichung $\mathbf{z}_n^{i}$ berechnet werden. Dann ist allerdings die Inversion der Jakobi-Matrix erforderlich.

Der nachstehende Algorithmus skizziert das Lösungsverfahren:

```
Festlegungen:    Schaltkreis: (C),
                 Eingangssignale: (E),
                 Zeitsteuerung: h, Tmax,

Ergebnisse:      Knotenpotentiale, Zweigströme: z(t).
Anmerkung:       n: Zeitschritt, i: Iterationsschritt.

BEGIN            Circuit-Analyse (C, E, h, tmax)
                 Lösung g(z) = 0 bei t = 0;  z0(1) = Anfangsbedingungen;
                 Auffüllen von A und b durch Linearisierung der
                 Modellgleichungen; Lösung von A * z0(i) = b ;

TimeLoop:        t = h ; n = 1 .
                 FOR t ≤ Tmax DO
                     BEGIN i = 1
                     FOR ABS (z(n,i)-(z(n,i-1)) < Eps DO
                         BEGIN: Auffüllen von A und b durch
                             Linearisierung der Modellgleichungen;
                             Lösung von A * zn(i) = b ;
                             i = i + 1 ;
                     END
                 t = t + h; n = n + 1;
                 END
END
```

Der Algorithmus zeigt im Inneren die Iterationsschleife mit dem Laufindex i und schließlich die Zeitschleife mit dem Laufindex n. Das Know-how steckt neben der geeigneten Linearisierung der Modellgleichungen insbesondere in der vorteilhaften Steuerung der Iterations- und Zeitschleife. Wichtig dabei ist eine adaptive Schrittweite mit Anpassung an die Änderungen in der Signalform. In Bild 5.1 ist der Ablauf der Circuit-Simulation in einem Ablaufdiagramm veranschaulicht. In der inneren Iterationsschleife ist ein nichtlineares algebraisches Gleichungssystem zu lösen. Bild 5.2 zeigt die iterative Lösung einer nichtlinearen Gleichung g(z) = 0 nach der Newton-Raphson Methode. Diese prinzipielle Darstellung zeigt die Komplexität eines Circuit-Simulators für eine zeitkontinuierliche Lösung. Daraus wird verständlich, daß mit dem Circuit-Simulator nur Schaltungen mit geringerer Komplexität analysiert werden können.

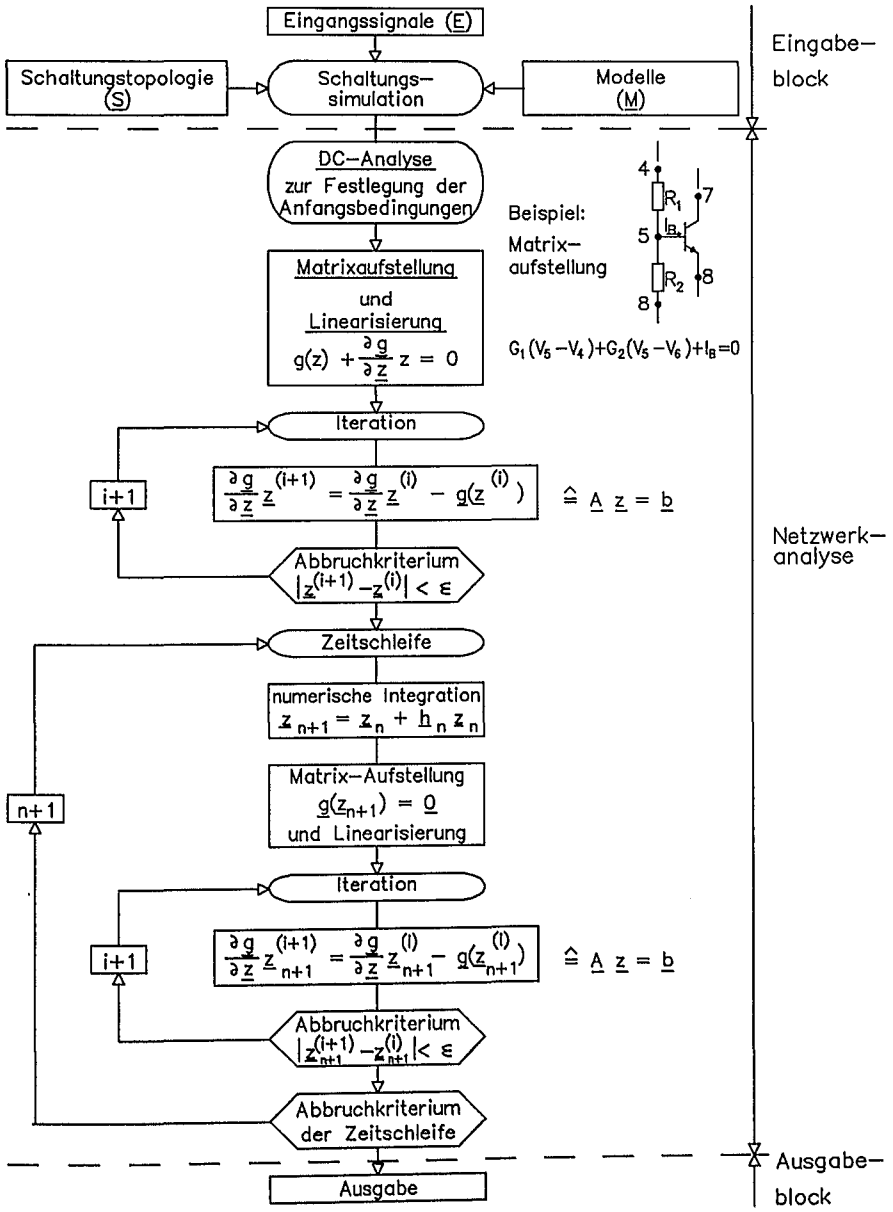


Bild 5.1: Zum Ablauf der Circuit–Simulation

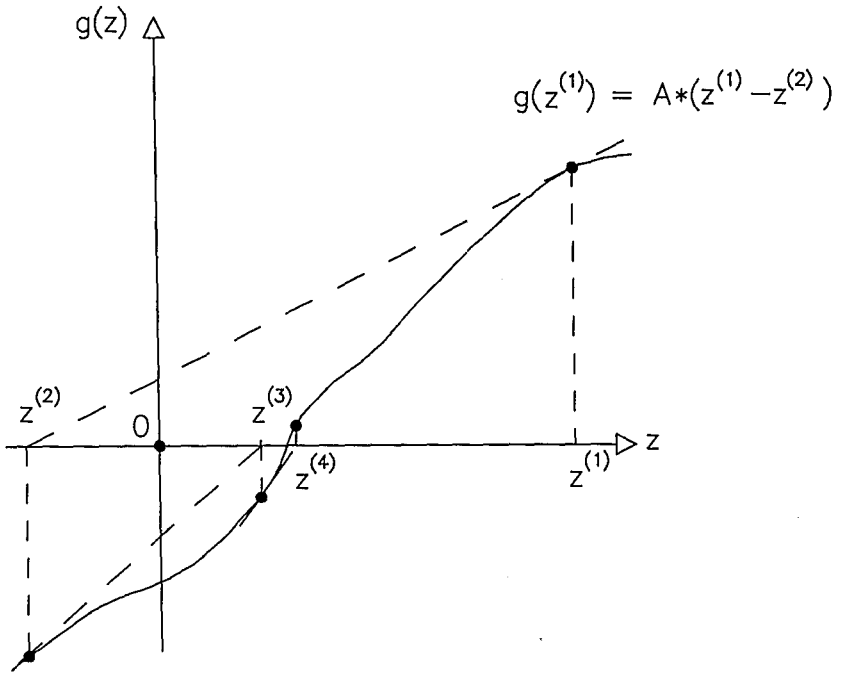


Bild 5.2: Zur Nullstellensuche nach Newton–Raphson

Zur Modellierung der Netzwerkelemente: Jedes Netzwerkelement wird durch Linearisierung und Zeitdiskretisierung in ein resistives Element der Form $G^*U_n = I_n$ zurückgeführt. In Bild 5.3 ist dies dargestellt für die Netzwerkelemente R, C, L, D. Die Diode D muß für den Zeitschritt n in jedem Iterationsschritt i durch ein linearisiertes Element mit $[g^{(i)}_{eq}; I^{(i)}_{eq}]$ beschrieben werden. In gleicher Weise ist dies auch für Transistorelemente erforderlich. Die Modelle für Bipolar–Transistoren und Feldeffekt–Transistoren sind um einiges aufwendiger, um das reale Verhalten hinreichend genau nachzubilden. Beispielsweise enthält das Gummel–Poon–Transistormodell für den Bipolar–Transistor ca. 40 Modellparameter, um Einflüsse wie z.B. Basisweitenmodulation, Hochstrominjektionseffekt, stromabhängige Stromverstärkung und stromabhängige Transferzeit der Ladungsträger zu beschreiben /9/.

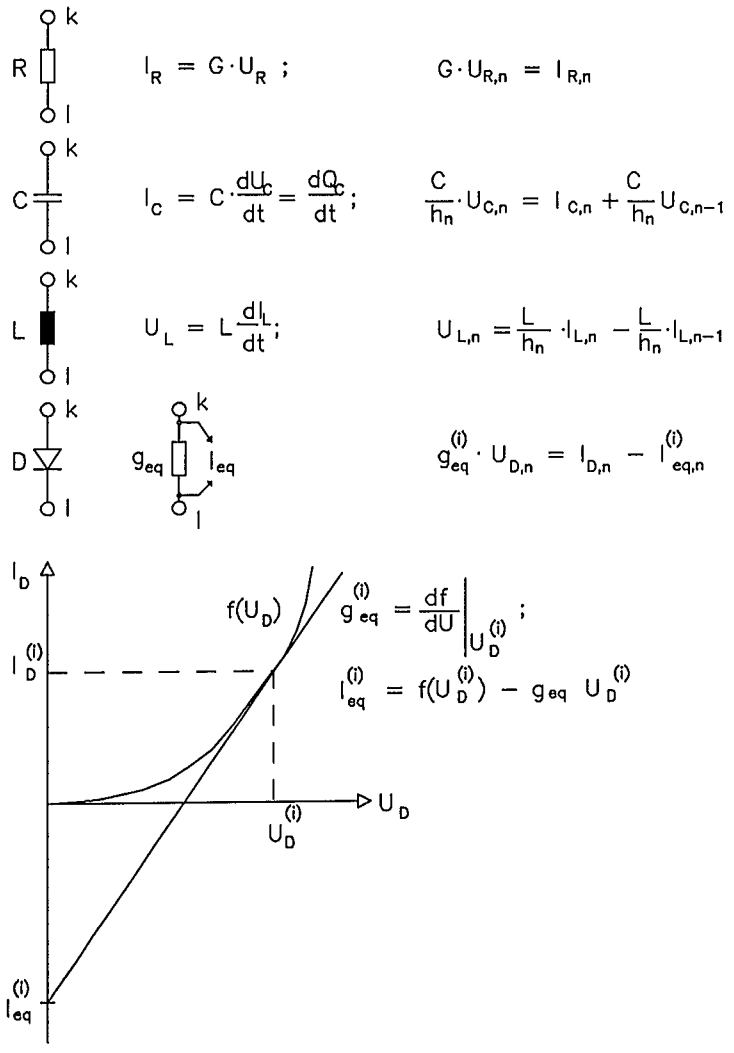


Bild 5.3: Zur Modellierung und Linearisierung der Netzwerkelemente

Zur Aufstellung der Netzwerkgleichungen: Die nichtlinearen Netzwerkgleichungen werden bei der MNA–Methode (Modified Nodal Analysis) nach der Knotenanalyse aufgestellt. Bild 5.4 verdeutlicht dies anhand eines einfachen Beispiels. Das Netzwerk wird dabei in Knoten und Zweige eingeteilt. Zugänglich für den Ausgabeblock sind die Knotenpotentiale und die Zweigströme.

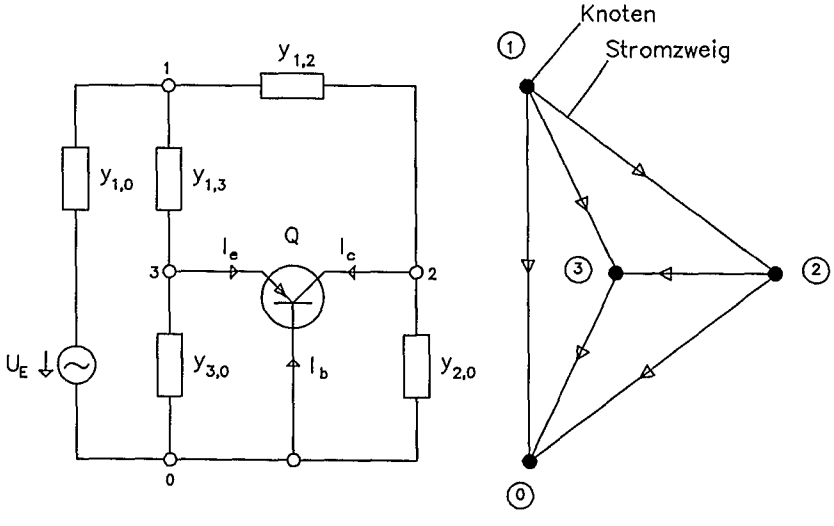


Bild 5.4: Beispiel für die Bildung der Knotenleitwertmatrix

Die geränderte Leitwertmatrix für den Bipolar–Transistor im Arbeitspunkt lautet:

$$\begin{bmatrix} I_b \\ I_c \\ I_e \end{bmatrix} = \begin{bmatrix} G_1 & S_r & -(G_1+S_r) \\ S & G_2 & -(S+G_2) \\ -(G_1+S) & -(S_r+G_2) & (S+S_r+G_1+G_2) \end{bmatrix} \cdot \begin{bmatrix} U_{b,0} \\ U_{c,0} \\ U_{e,0} \end{bmatrix} \qquad (5.6)$$

Die linearisierte Netzwerkgleichung zum Zeitpunkt n im Iterationsschritt i lautet für das einfache Beispiel in Bild 5.4:

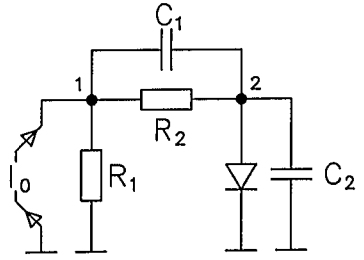
$$\begin{bmatrix} y_{1,0}+y_{1,3}+y_{1,2} & -y_{1,2} & -y_{1,3} \\ -y_{1,2} & y_{1,2}+y_{2,0}+G_2 & -(S+G_2) \\ -y_{1,3} & -(S_r+G_2) & y_{1,3}+y_{3,0}+(S+S_r+G_1+G_2) \end{bmatrix} \bullet \begin{bmatrix} U_{10} \\ U_{20} \\ U_{30} \end{bmatrix} = \begin{bmatrix} y_{1,0} \bullet U_E \\ 0 \\ 0 \end{bmatrix}$$

Diese Formulierung läßt sich auf folgende Form bringen:

$$(Y) \cdot (V) = (J) \qquad (5.7)$$

Dabei ist (V) ein Spaltenvektor mit allen Knotenpotentialen. Auf der rechten Seite stehen die Erregungen des Netzwerks. Wie man leicht sieht, ergeben sich schon Probleme bei $Z_{1,0} = 0$. Aus diesem Grund muß die Aufstellung der Netzwerkgleichungen modifiziert werden. Diese erforderliche Modifikation wird z.B. im MNA–Verfahren berücksichtigt.

Die Vorgehensweise beim Aufstellen der Netzwerkmatrix unter Anwendung des einfachen Knotenpotentialverfahrens bei Vorhandensein nichtlinearer Elemente verdeutlicht Bild 5.5. Gegenüber dem Beispiel in Bild 5.4 sind dort Energiespeicher und ein nichtlineares Netzwerkelement gegeben.



$$\begin{bmatrix} G_1+G_2+\frac{C_1}{h_n} & -G_2-\frac{C_1}{h_n} \\ -G_2-\frac{C_1}{h_n} & G_2+g_{eq}^{(i)}+\frac{C_1}{h_n}+\frac{C_2}{h_n} \end{bmatrix} \cdot \begin{bmatrix} V_1 \\ V_2 \end{bmatrix} = \begin{bmatrix} I_0+\frac{C_1}{h_n}V_{C_1,n-1} \\ -I_{eq}-\frac{C_1}{h_n}V_{C_1,n-1}+\frac{C_2}{h_n}V_{C_2,n-1} \end{bmatrix}$$

Bild 5.5: Knotenleitwertmatrix mit nichtlinearem Element

Die Nachteile des einfachen Knotenpotentialverfahrens sind u.a.:

* ohne zusätzliche Manipulationen ist der Einbau von unabhängigen Spannungsquellen, spannungsgesteuerten Spannungsquellen u.a. nicht möglich.

* die Induktivität führt bei DC zu einem unendlichen Leitwert.

Die Formulierung der Netzwerkgleichungen muß demzufolge modifiziert werden (z.B. MNA-Methode).

Zur MNA-Methode: Die MNA-Methode umgeht durch eine andere Formulierung der Netzwerkgleichungen die aufgezeigten Nachteile der einfachen Knotenanalyse. Die MNA-Methode ist sehr weit verbreitet, u.a. im Circuit-Simulator SPICE. Aufgebaut wird die Netzwerkmatrix nach folgendem Schema:

$$\begin{bmatrix} (Y) & (B) \\ (C) & (D) \end{bmatrix} * \begin{bmatrix} (V) \\ (I) \end{bmatrix} = \begin{bmatrix} (J) \\ (E) \end{bmatrix} \qquad (5.8)$$

Diese Formulierung stellt eine Erweiterung der Grundform (5.7) dar. Auf der rechten Seite stehen wieder die Erregungen in Form von Strömen (J) und Spannungen (E). (V) sind die Knotenpotentiale und (I) die Zweigströme. Grundsätzlich unterscheidet man zwischen den "Knotenpotentialgleichungen" und den "Zweigstromgleichungen" (Bild 5.6).

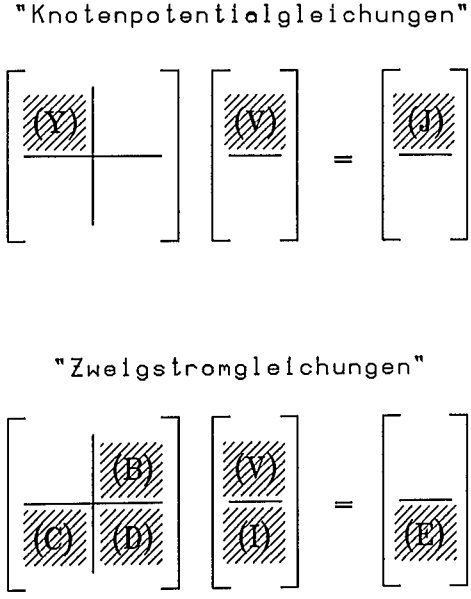
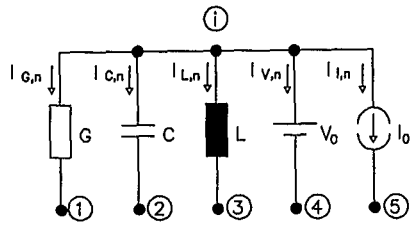


Bild 5.6: Zur Formulierung der MNA–Netzwerkmatrix

In Bild 5.7 ist dargestellt, wie bestimmte Netzwerkelemente in die Netzwerkmatrix Gl. 5.8 eingetragen werden. Für die Elemente R, C, J_0 ist ein Eintrag sowohl aller "Knotenpotentialgleichungen" als auch der "Zweigstromgleichungen" möglich. Bei den Netzwerkelementen für L und V_0 sind nur die "Zweigstromgleichungen" sinnvoll. Die Ströme (J) auf der rechten Seite bestimmen sich u.a. auch aus den bekannten Knotenpotentialen zum Zeitschritt n–1.



Element-typen	"Knotenpotentialgleichung" $\begin{bmatrix} \cdot & \cdot \\ \cdot & \cdot \end{bmatrix} \begin{bmatrix} V \\ I \end{bmatrix} = \begin{bmatrix} J \\ E \end{bmatrix}$	"Zweigstromgleichung" $\begin{bmatrix} \cdot & \cdot \\ \cdot & \cdot \end{bmatrix} \begin{bmatrix} V \\ I \end{bmatrix} = \begin{bmatrix} J \\ E \end{bmatrix}$
G	* $V_{i,n}$ $V_{1,n}$ RHS i G −G 1 −G G	$V_{i,n}$ $V_{1,n}$ $I_{G,n}$ RHS i 1 1 −1 BR G −G −1
C	* $V_{i,n}$ $V_{3,n}$ i $\frac{C}{h_n}$ $\frac{-C}{h_n}$ $\frac{C}{h_n} * V_{C,n-1}$ 2 $\frac{-C}{h_n}$ $\frac{C}{h_n}$ $\frac{-C}{h_n} * V_{C,n-1}$	$V_{i,n}$ $V_{2,n}$ $I_{C,n}$ RHS i 1 2 −1 BR $\frac{C}{h_n}$ $\frac{-C}{h_n}$ −1 $\frac{C}{h_n} * V_{C,n-1}$
L		$V_{i,n}$ $V_{3,n}$ $I_{L,n}$ RHS i 1 3 −1 BR 1 −1 $\frac{-L}{h_n}$ $\frac{-L}{h_n} * I_{L,n-1}$
V_0		$V_{i,n}$ $V_{4,n}$ $I_{V,n}$ RHS i 1 4 BR 1 −1 V_0
I_0	* V V RHS i $-I_0$ 5 I_0	$V_{i,n}$ $V_{5,n}$ $I_{I,n}$ RHS i 1 5 −1 BR 1 I_0

Bild 5.7: Eintrag der Netzwerkelemente bei der MNA-Methode
RHS : rechte Seite; BR : Zweigstromgleichung

Bei der Netzwerkanalyse wird eine DC–Analyse für das statische Verhalten einer Schaltung, eine AC–Analyse im Arbeitspunkt für das Verhalten im Frequenzbereich und eine transiente Analyse für das zeitkontinuierliche Verhalten eines nichtlinearen Schaltkreises durchgeführt. In Gleichung (5.9) bzw. (5.10) ist für das Beispiel in Bild 5.8 die MNA–Netzwerkmatrix dargestellt. Jede Zeile stellt eine Netzwerkgleichung dar; ein Zeilentausch ist somit jederzeit möglich. In Gleichung (5.10) ist gegenüber (5.9) die erste und vierte Zeile vertauscht.

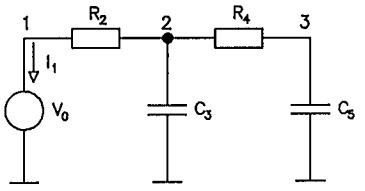


Bild 5.8: Zum Beispiel einer MNA–Matrix bei DC–Analyse

$$\begin{bmatrix} G_2 & -G_2 & 0 & 1 \\ -G_2 & G_2+G_4 & -G_4 & 0 \\ 0 & -G_4 & G_4 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} * \begin{bmatrix} V_1 \\ V_2 \\ V_3 \\ I_1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ V_0 \end{bmatrix} \qquad (5\text{–}9)$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ -G_2 & G_2+G_4 & -G_4 & 0 \\ 0 & -G_4 & G_4 & 0 \\ G_2 & -G_2 & 0 & 1 \end{bmatrix} * \begin{bmatrix} V_1 \\ V_2 \\ V_3 \\ I_1 \end{bmatrix} = \begin{bmatrix} V_0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \qquad (5\text{–}10)$$

Die AC–Analyse setzt eine DC–Analyse voraus. In der vorausgehenden DC–Analyse werden die Arbeitspunkte der Schaltungselemente bestimmt. Die AC–Analyse wird dann in den jeweiligen Arbeitspunkten durchgeführt. Die AC–Analyse ist eine Frequenzbereichsanalyse. Bei Linearisierung der Schaltungselemente in den Arbeitspunkten können die zeitlichen Ableitungen $\delta/\delta t$ durch $j\omega$ ersetzt werden. Dies vereinfacht die Analyse ganz erheblich. Aus Gleichung (5.11) ist die MNA–Netzwerkmatrix für eine AC–Analyse des Beispiels in Bild 5.9 zu entnehmen.

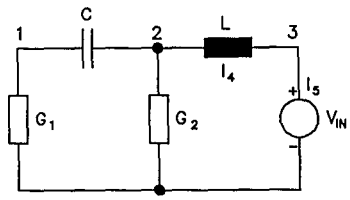


Bild 5.9: Zum Beispiel einer MNA-Matrix bei AC-Analyse

Gleichung 5.11 zum Beispiel nach Bild 5.9:

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} G_1+j\omega C & -j\omega C & 0 & 0 & 0 \\ -j\omega C & G_2+j\omega C & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & -1 & 1 & j\omega L & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix} \end{array} * \begin{bmatrix} V_1 \\ V_2 \\ V_3 \\ I_4 \\ I_5 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ V_{IN} \end{bmatrix} \qquad (5\text{-}11)$$

Bei der konkreten Realisierung eines Circuit-Simulators sind viele Detailprobleme zu lösen. Um einen Eindruck zu vermitteln, soll hier nur beispielhaft das Diagonalenproblem der MNA-Methode angesprochen werden. Nullstellen in der Hauptdiagonalen der Netzwerkmatrix führen zu Problemen bei der nach Gleichung (5.5) notwendigen Matrixinversion. Eine derartige Netzwerkmatrix nennt man singulär. Die Singularität kann durch Zeilentausch behoben werden, so wie in Gleichung (5.10) dargestellt. Man benötigt demnach einen Algorithmus, der die Behebung der singulären Stellen durch geeignete Zeilen- und Spaltentauschmanipulationen vornimmt. In der Praxis werden nicht die Zeilen getauscht, sondern es wird lediglich das Zeigersystem der Matrixeinträge geändert.

Grundsätzlich ist auch ein anderer Aufbau der Netzwerkmatrix als nach der MNA-Methode möglich. Zustandsdarstellungen führen als Zustandsgrößen die Ladung q und den magnetischen Fluß ϕ ein /12/. Damit lassen sich manche Probleme der MNA-Methode leichter überwinden. Allerdings handelt man sich an anderer Stelle wieder Nachteile ein.

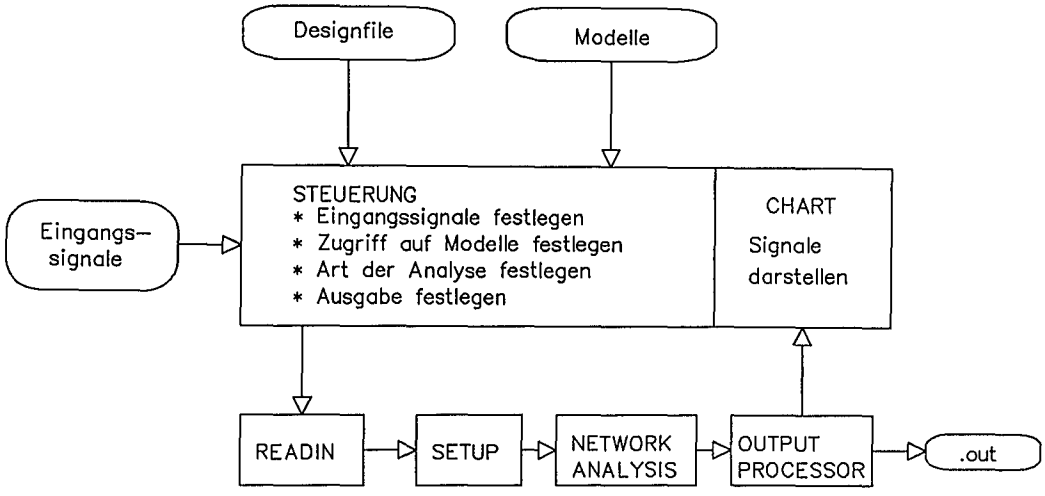
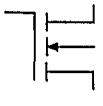


Bild 5.10: Aufbau eines Circuit–Simulators

Zum Aufbau eines Circuit–Simulators: Der grundsätzliche Aufbau eines Circuit–Simulators ist in Bild 5.10 dargestellt. Der Circuit–Simulator besteht aus einem READIN–Block zur Bereitstellung der Daten. Im SETUP–Block wird die Netzwerkmatrix vorbereitet; im NETWORK–ANALYSIS–Block schließlich werden die Netzwerkgleichungen aufgebaut und gelöst. Die Zwischenergebnisse für Knotenpotentiale und Zweigströme werden in einem Ergebnisspeicher abgelegt. Der OUTPUT–Prozessor bereitet die gewünschten Ergebnisparameter auf, die dann mit einem Graphikprogramm (z.B. CHART) graphisch dargestellt werden können. Gesteuert wird der Circuit–Simulator von einem Steuerblock. Dort erfolgen die Festlegungen für die Eingangssignalformen, die Art der Analyse, die Zugriffspfade auf Modellparametersätze für Schaltungselemente und die Festlegung der Ergebnisparameter. In Bild 5.11 sind Modellparametersätze für Transistorelemente dargestellt. Die Modellgleichungen für die Transistorelemente sind im SETUP– bzw. NETWORK–ANALYSIS–Block implementiert. Durch die Modellparameter können bauteilspezifische Merkmale und Eigenschaften eingestellt werden.

```
.MODEL QNL NPN(BF=80 RB=100 CCS=2PF TF=0.3NS TR=6NS CJE=3PF
+ CJC=2PF VA=50)
```



```
.MODEL N NMOS(VTO=0.4 KP=20E-6 GAMMA=1.5 PHI=1.5 LAMBDA=.016
+ CGSO=1.24E-9 CGDO=1.24E-9 CJ=2.55E-4 CJSW=2.55E-9 TOX=1E-7 XJ=2U)
```

Bild 5.11: Modellparametersätze für Transistorelemente

Um Circuit-Simulatoren schneller und effizienter zu machen, können abgesehen von Varianten in der Formulierung der Netzwerkmatrix, der Art der Zeitdiskretisierung und der Schrittweitensteuerung grundsätzlich folgende Maßnahmen getroffen werden:

* einfachere Modellierung, z.B. durch Makromodellbildung,

* Verzicht auf Genauigkeit der zeitkontinuierlichen Auflösung.

Genügt bei der zeitkontinuierlichen Auflösung eine 0/1/X-Darstellung der Signale, so können sehr viel einfachere Verfahren zur Analyse des Schaltungsverhaltens angewendet werden. Diese Eigenschaften der eingeschränkten Signaldarstellung werden in einem Logiksimulator voll ausgenutzt. Bild 5.12 veranschaulicht die Vereinfachungen in der Darstellungsform der Signale.

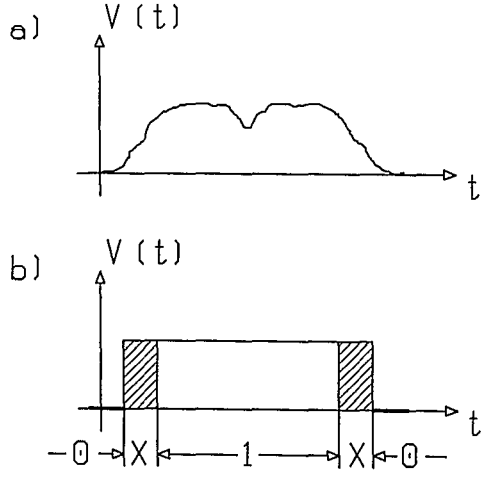


Bild 5.12: Zur Darstellungsform der Signale
a) zeitkontinuierliches Signal
b) 0-1-X-Darstellung des Signals

Switch-Level-Simulatoren: Diese sind zwischen den Circuit-Simulatoren und den Logiksimulatoren einzuordnen. Dabei wird ein MOS-Transistor durch einen gesteuerten Leitwert dargestellt, der nur Werte aus einer Menge vorgegebener diskreter Werte annehmen kann. Im einfachsten Fall wird der Transistor durch $R_{ON/OFF}$ dargestellt. Ein Switch-Level-Simulator ist ein angenäherter Circuit-Simulator. Der nachstehende Algorithmus verdeutlicht die Arbeitsweise:

```
Festlegungen:     Schaltkreis (C)
                  Zustände der Eingangsnetze E(NE)

Anmerkungen:      NE: Eingangsnetze
                  NI: Interne Netze
PROCEDURE         Q(NE, NI): Zustände der Transistorelemente
                  I(NI): Zustände an inneren Netzen


                  Transistor–Level–Simulation;
BEGIN             Update des Zustandes aller Transistorelemente Q(NE), die
                  mit Eingangsnetzen (NE) verbunden sind;
                  Q(NE): = E(NE);

                  Für alle internen Netze (NI)
                  REPEAT
                      BEGIN
                          Transistorelemente an internen Netzen(NI) im
                          Zustand Q(NI) festhalten;
                          Ermittle neuen Zustand I(NI) unter Anwen-
                          dung einer vereinfachten Netzwerkanalyse;
                          Transistorelemente an internen Netzen (NI)
                          auf neuen Zustand Q(NI) bringen;
                      END
                  UNTIL  bis keine Änderungen mehr auftreten

END
```

Switch–Level–Simulatoren wurden für die Analyse hochintegrierter digitaler MOS–Schaltungen entwickelt. Derartige Gatterstrukturen bestehen aus Subschaltungen, die nur schwach miteinander vernetzt sind. Jeder Knoten ist über Transistorelemente mit seinem Nachbarn verbunden. Hinzu kommen noch Kapazitäten gegen Masse bzw. zur Versorgungsspannung. Die Auswirkungen von einem Knoten zum Nachbarknoten lassen sich nacheinander berechnen, wobei jeweils nur der Nachbarknoten berücksichtigt werden muß.

Ein Eingangsereignis breitet sich gleichsam wie eine "Welle" in der Schaltung aus. Es werden all die Transistorelemente auf einen neuen Zustand gebracht, die von der Welle erfaßt werden. Jede Ereignisänderung muß solange abgearbeitet werden, bis alle Folgeereignisse in der Schaltung ermittelt sind. Bild 5.13 zeigt als Beispiel eine digitale Addierschaltung und den Signalflußgraphen der partitionierten Teilschaltungen. Aus diesem Graphen läßt sich die Fortpflanzung einer Ereignisänderung ausgehend vom Eingang ermitteln.

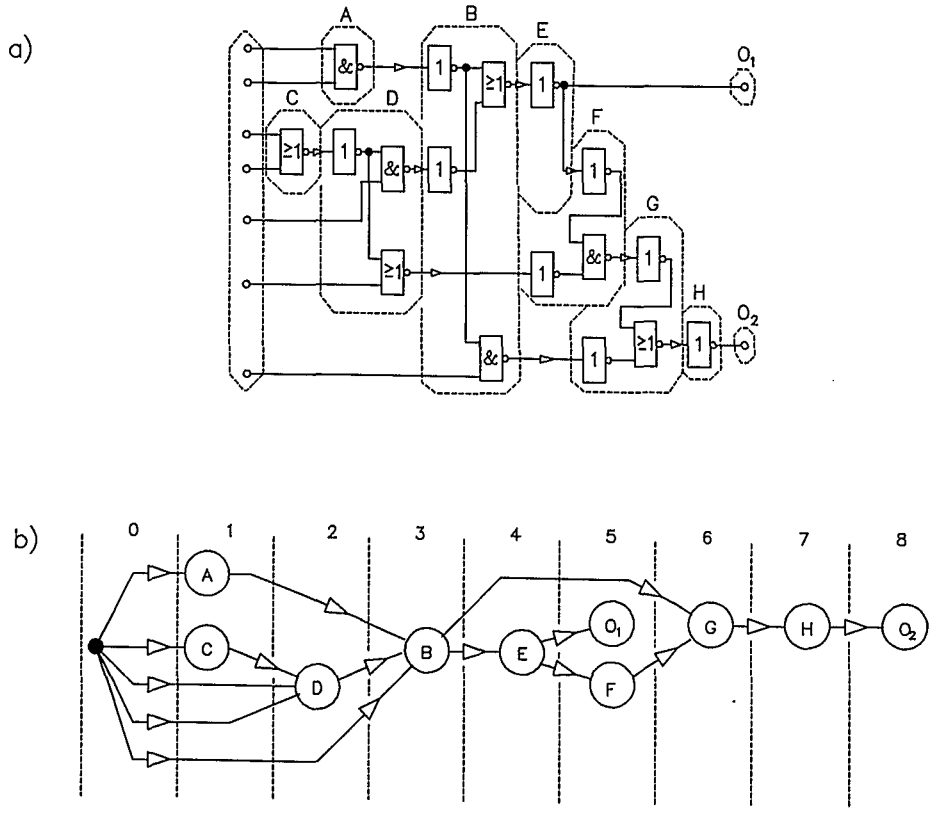


Bild 5.13: Zur Partitionierung von Netzwerken
a) Addierschaltung mit Partionierung des Netzwerkes
b) Signalflußgraph

Waveform–Relaxation–Methoden: Eine vorteilhafte Aufteilung einer Schaltung in Teilnetzwerke macht sich die WFR–Methode zunutze. Nach Bild 5.14 wird eine Schaltung in Teilnetzwerke aufgeteilt. Die Zerlegung in Teilnetzwerke erfolgt nach einem Algorithmus, der die Verkopplungen der Teilnetzwerke minimiert. Verkopplungen sollten möglichst lokal in den Teilnetzwerken vorliegen. Bei nicht verkoppelten Teilnetzwerken kann aus der Eingangserregung eines Teilnetzwerkes das Ausgangsverhalten z.B. nach der MNA–Methode berechnet werden. Die Ausgangssignale wirken wieder als Eingangssignale der nachfolgenden Teilnetzwerke u.s.w. Problematisch wird es bei Verkopplungen von Teilnetzwerken. Zunächst werden die Verkopplungen aufgetrennt (Bild 5.14). Anstelle der Rückkopplung wird am Eingang des betreffenden Teilnetzwerkes ein geschätzter Anfangssignalverlauf angenommen. Nach einem ersten Durchlauf wird der Schätzwert der Signalverläufe an den Rückkopplungsstellen verbessert. Nach mehreren Iterationen werden die zunächst angenommenen Signalverläufe sukzessive verbessert, bis ein geeignetes Abbruchkriterium den Iterationsvorgang beendet. Der nachstehende Algorithmus verdeutlicht die Vorgehensweise:

```
Festlegungen:    Schaltung: (C)
                 Eingangssignale: (E)
                 Zeitsteuerung: T_min; T_max
Ergebnisse:      Signalverläufe V(i, j, k, t)
                 i: Iterationsschritt
                 j: Subcircuit
                 k: Knoten eines Subcircuits

PROCEDURE        Waveform–Relaxation

BEGIN            Algorithmus zur Aufteilung der Schaltung (C) in
                 Teilnetzwerke (S);
                 Anzahl der Teilnetzwerke N_S;
                 V(0, *, *, t) = E;
                 i = 0
                 REPEAT
                     FOR j = 1 to N_S DO
                     BEGIN
                         Netzwerkanalyse für jedes Teilnetzwerk S(i, j);
                         Bestimme die Signalverläufe V(i, j, k, t);
                         Benutze bei der Iteration i die Ergebnisse der
                         Iteration i–1;
                     END
                 UNTIL |V(i, j, k, t) – V(i–1, j, k, t)| < ERROR
                 für alle j, k, t
END
```

Waveform–Relaxation–Algorithmus

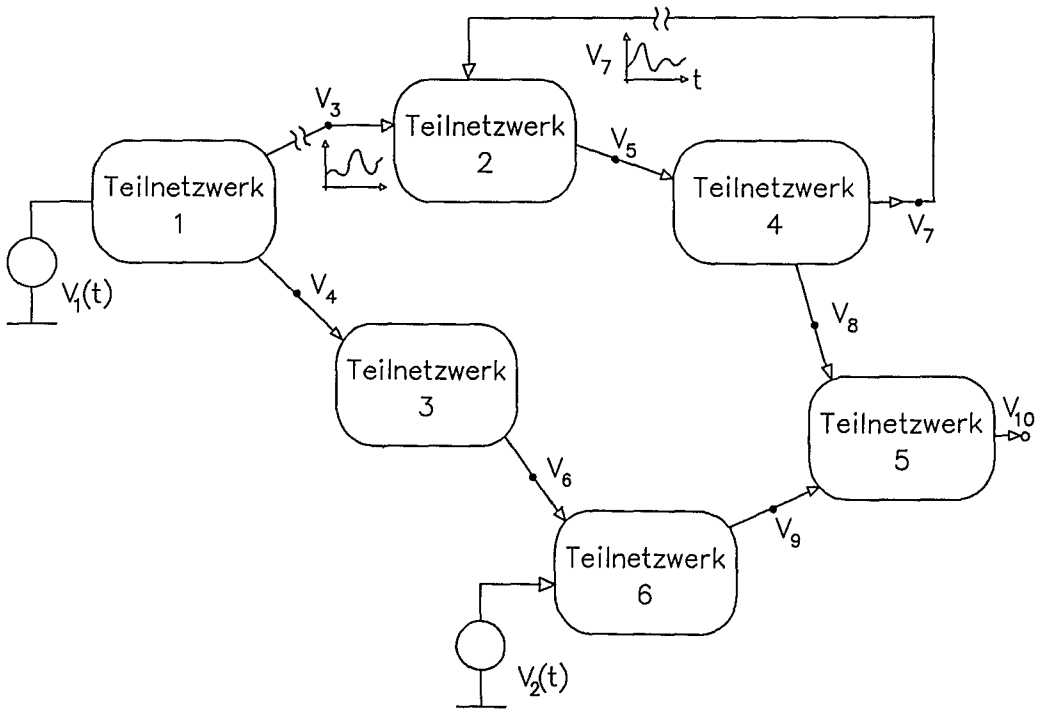


Bild 5.14: Zur WFR–Methode

5.2 Zur Methodik der Logik–Simulation

Bei der Logiksimulation werden die Signalverläufe einzelner Netze nicht zeitkontinuierlich dargestellt. Die Signale an den einzelnen Netzen können im einfachsten Fall nur die Werte 0/1/X annehmen. Jedes Schaltungselement wird durch das Logikverhalten beschrieben. Das Logikverhalten kann beschrieben werden durch:

* Boolesche Gleichungen,
* Wahrheitstabellen oder Funktionstabellen,
* Schaltung auf Gatterebene oder sonstige Schaltungsprimitive,
* Verhaltensmodell, z.B. mit VHDL–Prozedur,
* Meßergebnisse mit Hardware–Modellier–Einheit.

Aus dem so festgelegten Logikverhalten eines Schaltungselements erhält man die neuen Ausgangszustände für ein gegebenes Ereignis. Bild 5.15 zeigt, wie sich beispielsweise ein 0/1–Wechsel am Eingang IN1 auf die Ausgänge auswirkt.

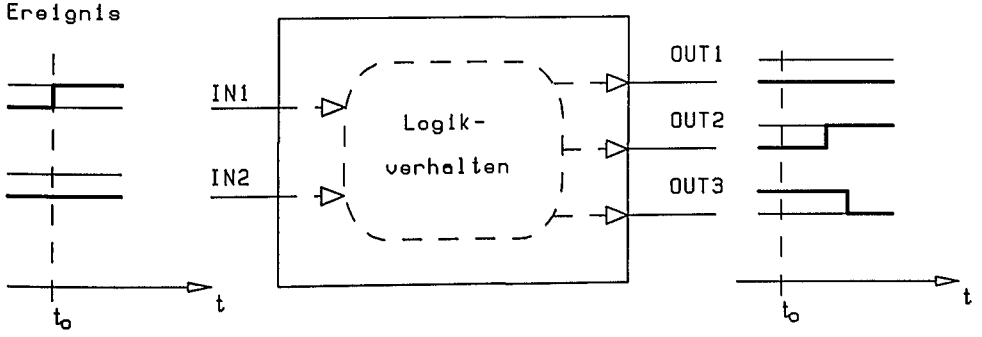


Bild 5.15: Zur Modellbildung von Schaltungselementen für die Logiksimulation

Die Fortpflanzung eines Ereignisses wird um innere Laufzeiten zusätzlich verzögert. Das zeitlich verzögerte Auftreten eines Ausgangsereignisses kann z.B. durch ein eigenes Timing–Modell beschrieben werden. Ein Problem ist, daß oftmals die Verzögerungszeiten u.a. von inneren Zuständen und von nebenläufigen Aktionen abhängig sind. Bild 5.16 zeigt ein Timing–Modell für ein einfaches NAND–Gatter. Im Timing–Modell ist die von Pin zu Pin wirksame minimale, typische und maximale Verzögerungszeit angegeben.

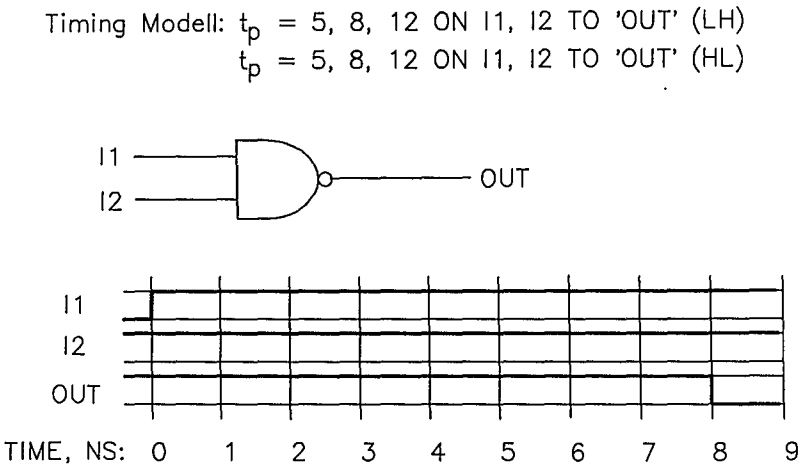


Bild 5.16: Zur verzögerten Auswirkung eines "Ereignisses"

Aufbau und Ablauf der Logiksimulation:

Inputs: Schaltung: **C**
Delays der Schaltelemente und Subcircuits: **ScD**,
Event–Queue: **EQ**,
Eingangs–Ereignisse: **IE**,
Results: Logikzustände von Netzen in Abhängigkeit von t.

```
PROCEDURE EventScheduling (C;ScD;t)
BEGIN
    EQ = IE ;
    WHILE EQ ist nicht leer DO
    BEGIN
        kleinster Zeitschritt t_n für nächste Ereignisänderung in EQ;
        P = alle Ereignisse von EQ zum Zeitpunkt t_n ;
        FOR all P(j) DO
        BEGIN
            S = Folgeereignisse von P(j);
            FOR all S(j) deren Zustand sich ändert DO
                Selective Tracing: Eintrag der Ereignisse
                S(i), t_n+ScD(i) in EQ;
        END
    END
END
```

Alle Ereignisse werden entsprechend ihres zeitlichen Auftretens in die Ereignisliste **EQ** eingetragen. Zum Startzeitpunkt erfolgt zunächst die Übernahme aller von den Eingängen herrührenden Eingangsereignisse **IE** in die Ereignisliste **EQ**. Ein Eingangsereignis wirkt auf die Schaltung und verursacht Folgeereignisse **S**. Die Folgeereignisse **S**(i) werden wiederum entsprechend ihres zeitlichen Auftretens t_n + **ScD**(i) in die Ereignisliste **EQ** eingetragen. Die Ereignisliste **EQ** wird solange abgearbeitet, bis die Schaltung "ruhig" ist. In **P** werden alle Ereignisse von **EQ** zum Zeitpunkt t_n zwischengespeichert, um so über den Subcircuitprozessor die Folgeereignisse zu ermitteln. In Bild 5.17 ist der prinzipielle Aufbau eines Logiksimulators skizziert.

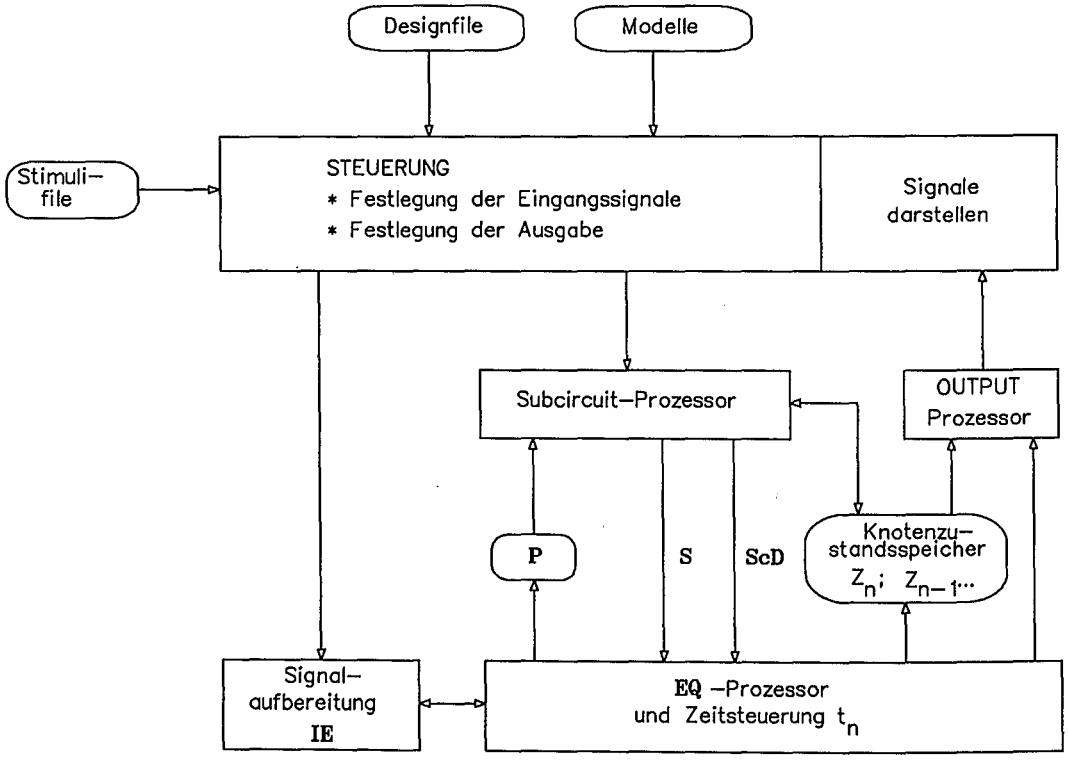


Bild 5.17: Zum Aufbau eines Logiksimulators

Ähnlich wie bei der Circuit–Simulation (Bild 5.10) wird auch der Logiksimulator von einem Steuerblock gesteuert. Dort erfolgt u.a. die Bereitstellung der Eingangssignale und die Bereitstellung der Modelle. Die zu simulierende Schaltung ist im Designfile abgelegt. Im Knotenzustandsspeicher werden bis zu einer einstellbaren "Speichertiefe" die Signalzustände aller Knoten abgelegt. Während vom Output–Prozessor nur die Signale verarbeitet werden, die bei der Festlegung der Ausgabe angegeben wurden.

5.3 Mixed–Mode–Simulation

Bei gemischt analog/digitalen Schaltungen versagt die reine Logiksimulation, und die Circuit–Simulation ist zu aufwendig. Für diesen Anwendungsbereich wurden Mixed–Mode–Simulatoren entwickelt. Die Gesamtschaltung wird partitioniert in analoge und digitale Teilschaltungen. Die analogen Teilschaltungen werden mit dem Circuit–Simulator, die digitalen Teilschaltungen mit dem Logiksimulator behandelt. An den analog/digitalen Schnittstellen erfolgt eine Signalumsetzung. Das analoge Signal wird nach Bild 5.12 in ein Digitalsignal umgesetzt. Die Ausgänge des Logiksimulators werden als Eingangssignale für den Circuit–Simulator aufbereitet. Den prinzipiellen Aufbau eines Mixed–Mode–Simulators zeigt Bild 5.18.

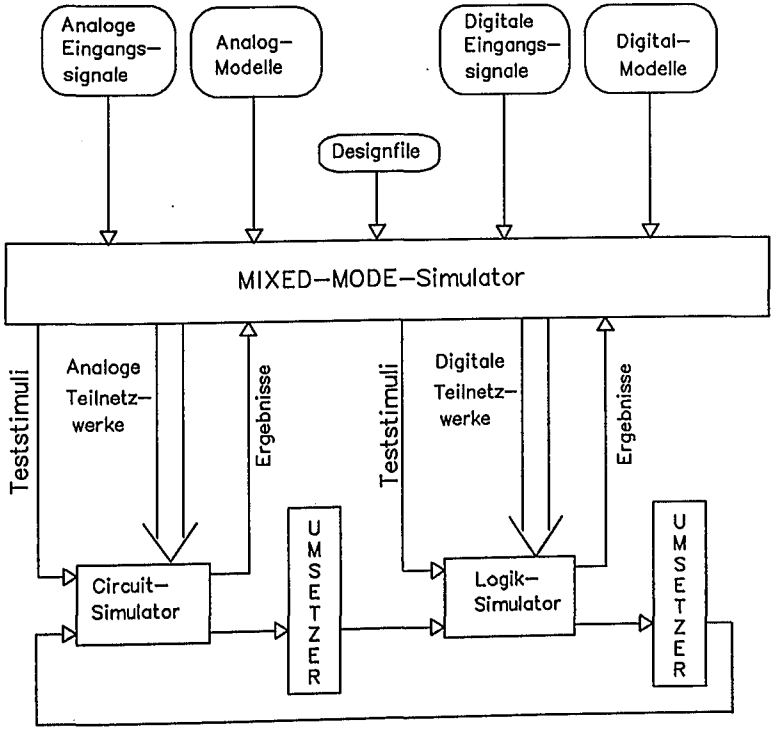


Bild 5.18: Zum Grundprinzip der Mixed–Mode–Simulation

5.4 Strukturabhängige Simulation

Bei der Entwicklung elektronischer Systeme wird vor dem Aufbau das Systemverhalten durch Simulation verifiziert. Je aufwendiger und kostspieliger die technologische Realisierung ist, um so dringender und notwendiger ist die Verifikation durch Simulation. Dabei muß bei höheren Signalverarbeitungsgeschwindigkeiten aber auch der Einfluß der Aufbau- und Verbindungstechnik mit berücksichtigt werden. Bei der Logiksimulation ermittelt beispielsweise die ADD_DELAY-Funktion die zusätzlichen Kapazitäten der Verbindungsleitungen (Metallisierung) zwischen dem Ausgang und den Gate-Kapazitäten der Folgetransistoren.

Mit steigender Aufbaudichte in integrierten Systemen ergeben sich zunehmend Probleme durch elektrische und thermische Verkopplungen /10/. Circuit-Simulatoren müssen erweitert werden, um Laufzeiteffekte, das Übersprechverhalten, Signalreflexionen und Signalverformungen, verursacht durch die Aufbau- und Verbindungstechnik von Schaltkreisen, berücksichtigen zu können. Notwendig und erforderlich ist eine strukturabhängige Simulation. Bild 5.19 zeigt eine Verbindungsstruktur mehrerer Schaltkreiszellen. Durch die Verbindungsstruktur können u.a. Störimpulse durch Übersprechen entstehen. Möglich sind auch Signalverformungen durch Signalreflexionen.

Eine komplexe Verbindungsstruktur kann im allgemeinen in modellierbare Segmente zerlegt werden. Bild 5.19 zeigt die Zerlegung einer als kritisch gekennzeichneten Verbindungsstruktur in modellierbare Grundelmente. Die Grundelemente werden als Strukturelemente bezeichnet. Die Strukturelemente lassen sich in drei Gruppen einteilen:

* Leitungsgebilde (z.B. Mehrlagen-Mehrleiter-Element);
* Knotengebilde (z.B. Leitungskreuzung, Durchkontaktierung);
* Anschlußgebilde (z.B. Bondverbindung).

Die Leitungsgebilde können u.a. gestreckt, kreisförmig gebogen, oder um 90 Grad abknickend geformt sein. Über Knotengebilde läßt sich einem Verbindungsknoten ein elektrischer Einfluß zuordnen.

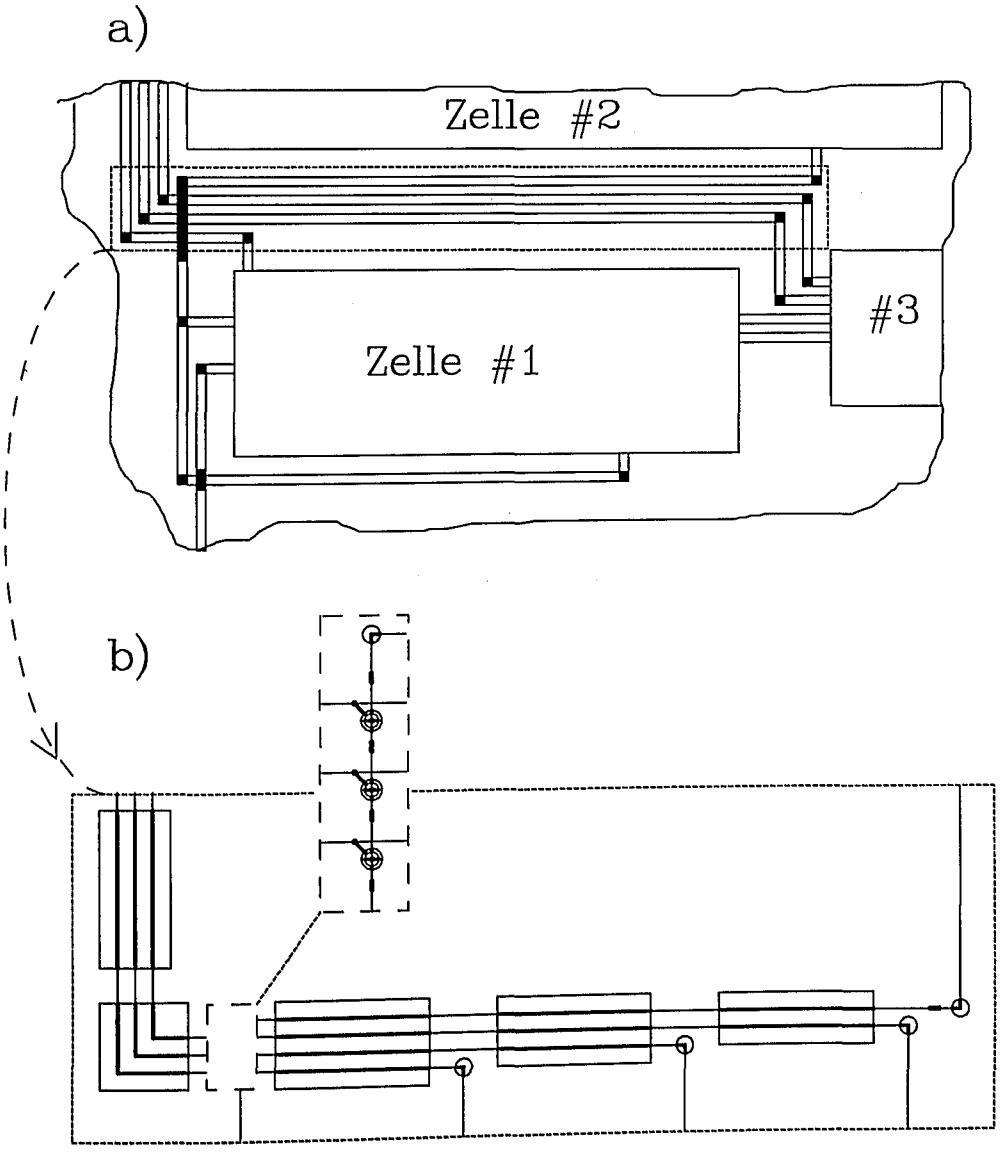


Bild 5.19: Zerlegung einer Verbindungsstruktur in modellierbare Segmente
a) Verbindungsstruktur von Schaltkreisen
b) Zerlegung in Segmente

Jedes Schaltungselement kann symbolisch, elektrisch oder geometrisch beschrieben werden. Bild 5.20 zeigt die Ergänzung der drei möglichen Beschreibungsarten um die Beschreibung von Strukturelementen. In Bild 5.19 werden beispielsweise die Segmente für Mehrlagen–Mehrleitergebilde durch geeignete Symbole dargestellt.

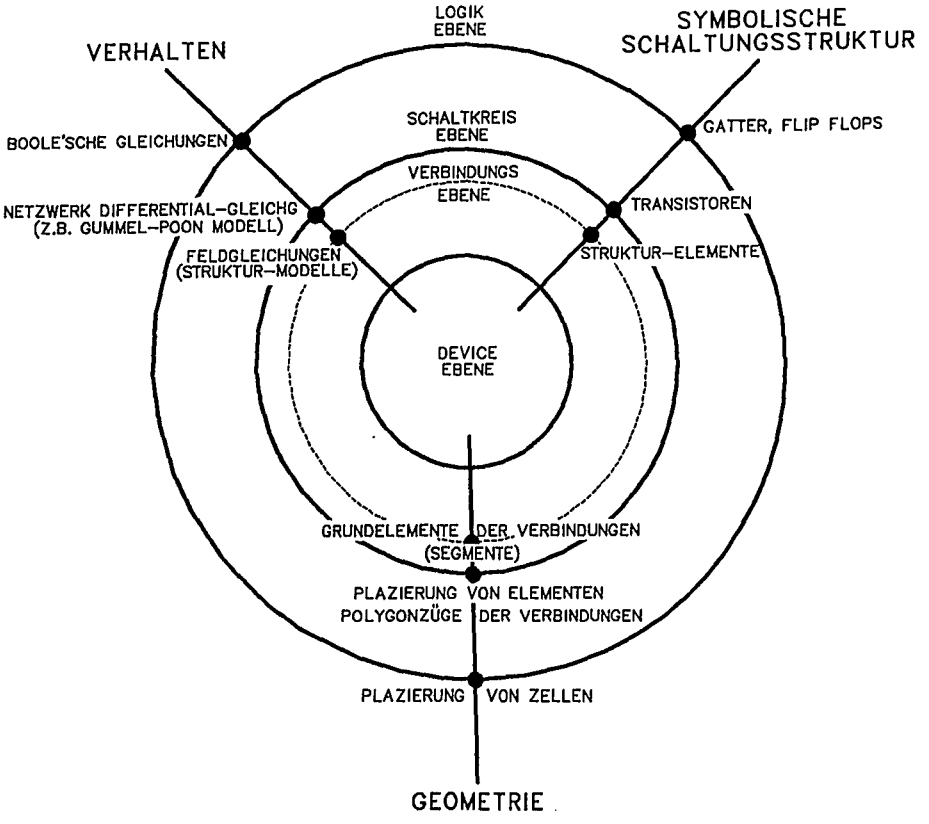


Bild 5.20: Zur Beschreibung von Strukturelementen

Wie diese drei Beschreibungsarten von Strukturelementen aussehen können, zeigt Bild 5.21 für ein Mehrlagen-Mehrleiter-Segment. Das Verbindungs-Segment wird in der grafischen Schaltplaneingabe durch ein Symbol dargestellt; im Layoutbereich ist es ein "Bauteil". Durch zusätzliche Werkzeuge auf einer CAE-Workstation kann aus dem Symbol bzw. aus dem "Bauteil" für das Verbindungs-Segment ein Modell generiert werden /11/. Mit den Modellen für Verbindungs-Segmente ist eine Simulation unter Berücksichtigung der Verbindungsstruktur möglich.

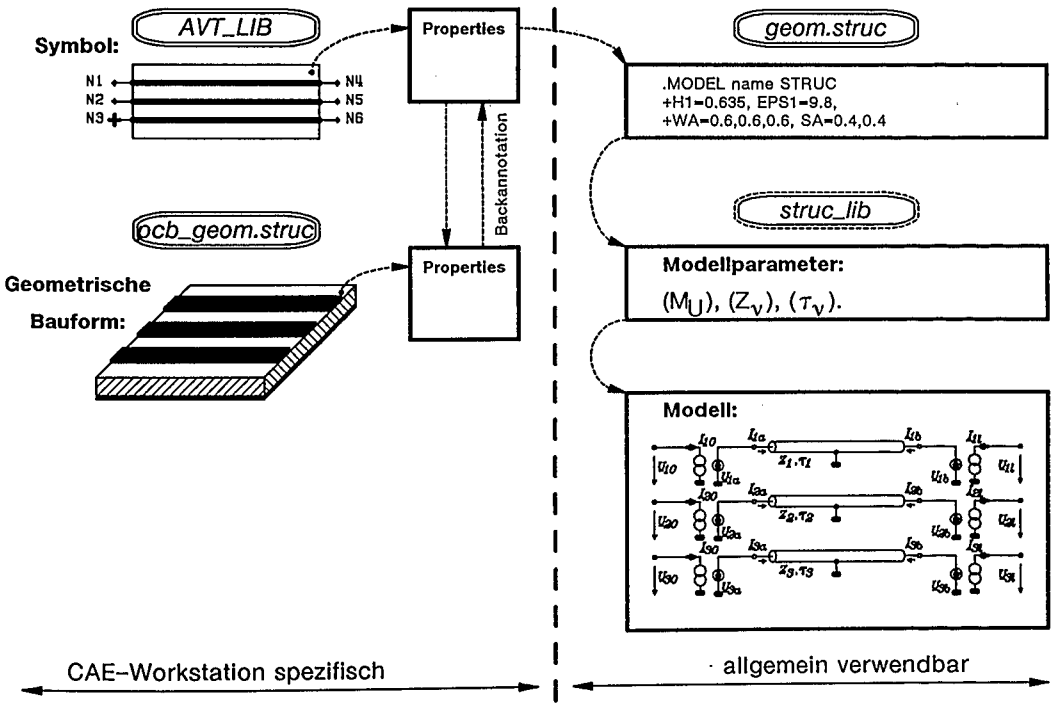


Bild 5.21: Zur symbolischen, geometrischen und elektrischen Darstellung eines Verbindungs-Segments

Literatur zu Kapitel 5

/1/ L. O. Chua, P. M. Lin: Computer-Aided Analysis of Electronic Circuits. Englewood Cliffs, NJ: Prentice-Hall, 1975

/2/ D. A. Calahan: Rechnergestützter Schaltungsentwurf. München: Oldenburg, 1973

/3/ E.-H. Horneber: Simulation elektrischer Schaltungen auf dem Rechner. Berlin: Springer, 1980

/4/ F.-Th. Mellert: Rechnergestützter Entwurf elektrischer Schaltungen. München: Oldenburg, 1981

/5/ A. Vladimirescu et al.: SPICE Version 2G User's Guide. Department of Electrical and Computer Science; University of California, Berkeley, 1981

/6/ F. H. Brannin: Computer Methods of Network Analysis. IEEE Proceedings, Vol.55, Heft 11, 1967

/7/ D. O. Pederson: A Historical Review of Circuit Simulation. IEEE Transactions on Circuits and Systems, Vol. CAS- 31, Heft 1, 1984

/8/ A. E. Rühli, G. S. Ditlow: Circuit Analysis, Logic Simulation and Design Verification for VLSI. IEEE Proceedings, Vol. 71, Heft 1, 1983

/9/ I. Getreu: Modelling the Bipolar Transistor. Elsevier Scientific Publishing Company, Amsterdam-Oxford-New York, 1978

/10/ J. Siegl: Simulation der Aufbau- und Verbindungstechnik. Fachbeilage Mikroperipherik, Mikroelektronik, Band 4, Heft 2, 1990

/11/ J. Siegl, K.H. Wirth, R. Michelfeit: Erweiterung eines CAE-Systems zur Schaltkreissimulation unter Berücksichtigung der Aufbau- und Verbindungstechnik. Frequenz, 1990

/12/ R. Schwarz: Analyse nichtlinearer Netzwerke im erweiterten Zustandsraum. München: Oldenburg, 1989

Fragen zum 5. Kapitel

F5.1 Skizzieren Sie den inneren Ablauf eines Circuit–Simulators bei einer Zeitbereichsanalyse.

F5.2 Skizzieren Sie den Aufbau der Netzwerkmatrix beim Knotenpotential–verfahren.

F5.3 Welche wesentlichen Modifikationen werden bei der MNA–Methode gegenüber dem Knotenpotentialverfahren vorgenommen und wie ist die MNA–Netzwerkmatrix aufgebaut?

F5.4 Wie erfolgt die Linearisierung des nichtlinearen Elements Diode; mit welchen Parametern wird die Ersatzschaltung beschrieben?

F5.5 Skizzieren Sie die Nullstellensuche nach dem Newton–Raphson–Verfahren zur Lösung eines nichtlinearen Netzwerkproblems.

F5.6 Skizzieren Sie den prinzipiellen Aufbau eines Circuit–Simulators; in welche Blockfunktionen kann ein Circuit–Simulator gegliedert werden?

F5.7 Wie arbeitet ein Switch–Level–Simulator und wofür wird er hauptsächlich verwendet?

F5.8 Erläutern Sie die Waveform–Relaxation–Methode; welche Vor– und Nachteile beinhaltet sie?

F5.9 Wie arbeitet ein Logiksimulator; skizzieren sie den Ablauf bei einer Logiksimulation.

F5.10 Wie ist prinzipiell ein Logiksimulator aufgebaut?

F5.11 Wie ist ein Mixed–Mode–Simulator aufgebaut und in welchen Bereichen wird er eingesetzt?

F5.12 Auf welche Art und Weise lassen sich Verbindungsstrukturen bei der Circuit–Simulation berücksichtigen?

6. Werkzeuge zur Layouterstellung

Das *Layout* ist die genaue Geometrie aller Strukturen auf einem Chip. Es ist das Ergebnis eines Chip–Entwurfs, aus dem die Maskensätze für die einzelnen Schritte des Herstellungsprozesses abgeleitet werden. Letztlich ist das Layout bzw. der Maskensatz die Schnittstelle zwischen der Entwurfsphase und der Fertigungsphase eines integrierten Schaltkreises. Bei Gate–Arrays liegt insofern eine besondere Situation vor, als die meisten Herstellungsschritte designunabhängig sind. Lediglich einige wenige kundenspezifische Masken– bzw. Herstellungsschritte werden zusätzlich benötigt. Um das Einhalten der geometrischen Entwurfsregeln, z.B. der Mindestabstände oder der Mindestüberlappungen zu garantieren, werden *symbolische interaktive* und *automatische* Layoutmethoden angewandt, bei denen bereits vorentwickelte Zellen, deren Layout in einer Bibliothek abgelegt ist, auf der nutzbaren Siliziumoberfläche plaziert werden. Für Zellen beliebiger Größe und Form ist dies sehr komplex und wird noch nicht allgemein beherrscht. Für rechteckige Zellen gleicher oder ähnlicher Größe, wie bei Gate–Arrays, oder gleicher Höhe, wie bei Standardzellen, existieren praktikable Algorithmen, so daß der Schaltungsentwickler sich im wesentlichen auf den Schaltungsentwurf konzentrieren kann. Die folgenden Ausführungen beschäftigen sich ausschließlich mit der Layouterstellung von *channeled* Gate–Arrays, um den Rahmen nicht zu sprengen.

6.1 Floorplanning und Plazierung

Floorplanning und Plazierung bestimmen die Position der Zellen im Layout. Von Plazierung wird gesprochen, wenn Zellen vorgegebener Größe und Gestalt verwendet werden. Dies gilt insbesondere für channeled Gate–Arrays. Floorplanning ist die Verallgemeinerung der Plazierung auf Zellen variabler Gestalt und bezeichnet die Nachbarschaftsrelation dieser Zellen. Darunter wird verstanden, ob die Zellen übereinander oder seitlich voneinander plaziert sind. Floorplanning wird u. a. auch bei *Sea Of Gates* ASIC angewandt.

Das Ziel von Floorplanning und Plazierung ist, eine vollständige Verdrahtung bei kleinster Fläche unter Berücksichtigung weiterer Randbedingungen zu erreichen. Ein Beispiel solcher zusätzlicher Randbedingungen wäre die Minimierung der totalen Netzlänge oder auch der Zahl der Netze, die eine gewisse Linie schneiden. Beide Größen sind eigentlich erst nach einer vollständigen Verdrahtung bekannt. Deshalb müssen praktikable Näherungen bzw. Abschätzungen als Optimierungsbasis verwendet werden. Dabei werden in der Regel rechtwinklige Verdrahtungen zugrundegelegt (Manhattan–Geometrie). Die kürzeste Verbindung kann in diesem Sinne als minimaler Steiner–Baum (siehe Bild 6.1a), als aufspannender Baum (siehe Bild 6.1b) oder als Kette (siehe Bild 6.1c) der zu verbindenden Punkte abgeschätzt werden. Eine noch gröbere Näherung stellt die *Methode des halben Umfangs* dar. Dabei wird der halbe Umfang des kleinsten Rechtecks, das die zu verbindenden Punkte einschließt, als geschätzte Netzlänge verwendet.

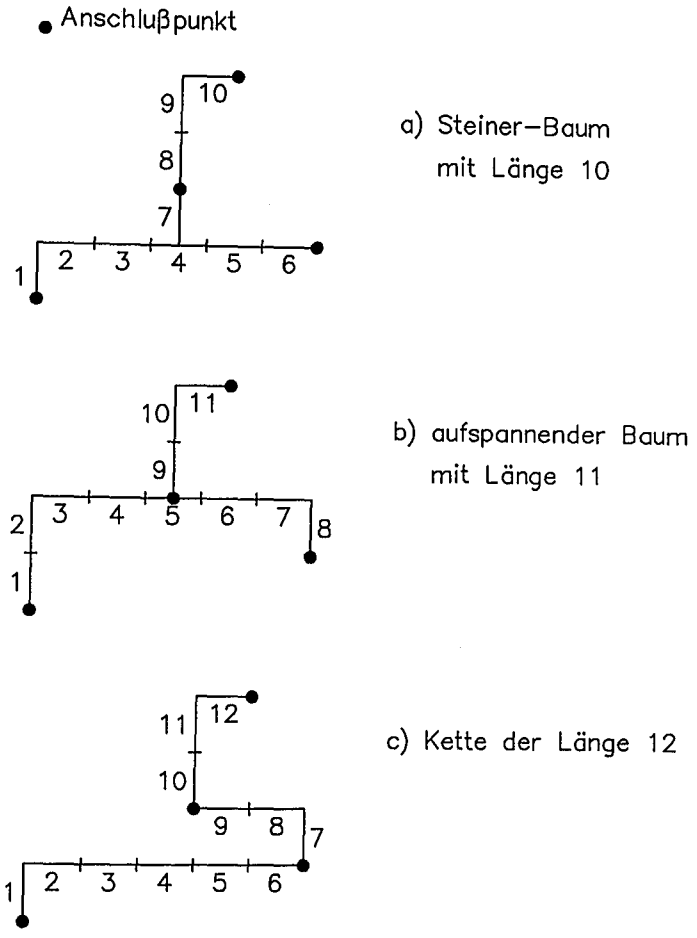


Bild 6.1: Verschiedene Ausführungen von Netzen

Andere Floorplanning– bzw. Plazierungsalgorithmen verwenden die *Min–Cut* Methode, *Clustering, Simulated Annealing* oder *kräftegesteuerte* Methoden.

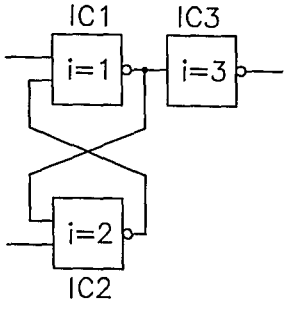
Beim *Cluster–Verfahren* wird die Chipfläche in Rechtecke (Partitionen, Bins) zerlegt und den Partitionen werden Zellen so zugeordnet, daß eng verkoppelte Zellen möglichst in eine Partition fallen. Dazu wird für jedes Zellen– oder Clusterpaar ein Cluster–Wert berechnet. Das Paar mit dem höchsten Clusterwert wird zu einem Cluster zusammengefaßt. Anschließend werden die Clusterwerte erneut berechnet. Der Clusterwert CV_{ij} ist dabei proportional zur Zahl der Kanten (Verbindungsnetze) zwischen den beiden Paarelementen dividiert durch die Zahl der Kanten, die nicht Teil der Paarverbindungen sind (siehe Bild 6.2 und Bild 6.3). Die

Clusterfunktion f(S) ist ein Gewichtsfaktor (häufig =1/S), der den Clusterwert so beeinflußt, daß die Größe S der Cluster, das ist üblicherweise die Zahl der Zellen und Zusammenfassungen von Paaren zu einem größeren Cluster, ausgeglichen bleibt.

$$CV_{ij} = f(s_j)\frac{C_{ij}}{T_j - C_{ij}} + f(s_i)\frac{C_{ij}}{T_i - C_{ij}} \; ; \; i \neq j$$

CV :	Clusterwert
C_{ij} :	Zahl der Kanten zwischen den Paarelementen
T_i ; T_j :	Zahl der Kanten an der Zelle i bzw. j des Zellpaares
$f(s_i)$;$f(s_j)$:	Cluster–Funktion; Gewichtsfaktor in Abhängigkeit der Clustergröße s (Zahl der Zellen im Cluster i bzw. j) (üblicherweise: f(s) = 1/s)

Bild 6.2: Eine einfache Methode zur Berechnung eines Clusterwerts



$C_{12}=2$; $T_1=3$
$C_{13}=1$; $T_2=3$
$C_{23}=1$; $T_3=2$

$$CV_{12} = 1*\frac{2}{3-2} + 1*\frac{2}{3-2} = 4$$

$$CV_{13} = 1*\frac{1}{3-1} + 1*\frac{1}{2-1} = 0{,}8 = CV_{23}$$

IC_1 und IC_2 werden zum Cluster zusammengefaßt.

Bild 6.3: Beispiel zur Berechnung eines Cluster–Werts und Bildung eines Clusters

Bei der *Min–Cut Methode* wird die Chipfläche sukzessive durch eine Trennlinie geteilt und Zellen bzw. Zellgruppen werden so ausgetauscht, daß die Zahl der Verbindungslinien, die von der Trennlinie geschnitten werden, reduziert wird.

Kräftegesteuerte Plazierungsmethoden basieren auf einer physikalischen Analogie. Je mehr eine Feder gedehnt wird, desto höher wird die Zugkraft der Feder. Im Rahmen dieser Methode wird eine Kraft zwischen zwei Zellen eingeführt, die proportional zu ihrer Entfernung und der Zahl der sie verbindenden Netze ist. Die optimale Plazierung ergibt sich, wenn die Summe aller Kräfte verschwindet. Dies ist natürlich der Fall, wenn der Abstand der Zellen verschwindet. Deshalb müssen für dieses Verfahren Platzvorgaben für einige Zellen, üblicherweise die I/O–Zellen, getroffen werden. Alternativ können zusätzliche kurzreichweitige abstoßende Kräfte eingeführt werden.

Wird nicht die Kraft direkt minimiert sondern die potentielle Energie, so spricht man von der Eigenwert–Methode. Die potentielle Energie ist die Summe der Abstandsquadrate, multipliziert mit der Zahl der Verbindungen zweier Zellen:

$$K = \Sigma\, c_{ij}\, \{(x_i - x_j)^2 + (y_i - y_j)^2\} = K_x + K_y$$

Durch Bestimmung der kleinsten, nicht verschwindenden Eigenwerte der Koeffizientenmatrix können optimale Zellpositionen ermittelt werden.

Ein weiteres Verfahren, das auf einer physikalischen Analogie basiert, ist die Methode des *Simulated Annealing*. Diese Methode entspricht der Abkühlung einer Flüssigkeit, die dabei langsam erstarrt. Die Moleküle der Flüssigkeit nehmen dabei Lagen mit näherungsweise niedrigster Energie ein. Beim langsamen Abkühlen kann ein Kristall, d. h. eine reguläre Anordnung mit minimaler Energie der Bausteine entstehen. Dabei spielt die thermische Bewegung der Bausteine während der Abkühlung eine wesentliche Rolle. Dieses physikalische Phänomen wird folgendermaßen für Plazierungszwecke simuliert: ausgehend von einer erstmaligen Zufallsplazierung wird eine neue Plazierung gewählt. Für die neue Plazierung wird ein Kostenfaktor K nach den betrachteten Optimierungskriterien berechnet. Falls der neue Kostenfaktor günstiger ist, wird die neue Plazierung als neue Ausgangsplazierung übernommen (physikalisches Analogon: geringere Gesamtenergie), andernfalls wird die neue Plazierung mit der Wahrscheinlichkeit $\exp(-\delta K/T)$ übernommen, d.h. wenn eine Zufallszahl zwischen 0 und 1 kleiner ist als $\exp(-\delta K/T)$. δK ist die Zunahme des Kostenkriteriums (physikalische Analogie: Zunahme der Gesamtenergie). T entspricht der Temperatur im physikalischen Analogon. Je höher die "Temperatur" T ist, desto höher wird auch die Wahrscheinlichkeit, daß temporär auch ungünstigere Konstellationen akzeptiert werden. Durch Variation dieser "Annealing–Temperatur", kann auf einfache Weise die Plazierungsgüte beeinflußt werden.

Das Layout–Werkzeug GATEPLACE von Mentor–Graphics führt erst eine globale Plazierung unter Anwendung der Cluster–Methode durch (siehe Bild 6.4).

Anschließend wird eine detaillierte Plazierung ausgeführt, bei der die einzelnen Makrozellen auf gültige Plätze gesetzt werden (siehe Bild 6.5). Die so vorplazierten Zellen werden noch innerhalb eines Bins vertauscht, um möglicherweise die Netzlängen zu reduzieren.

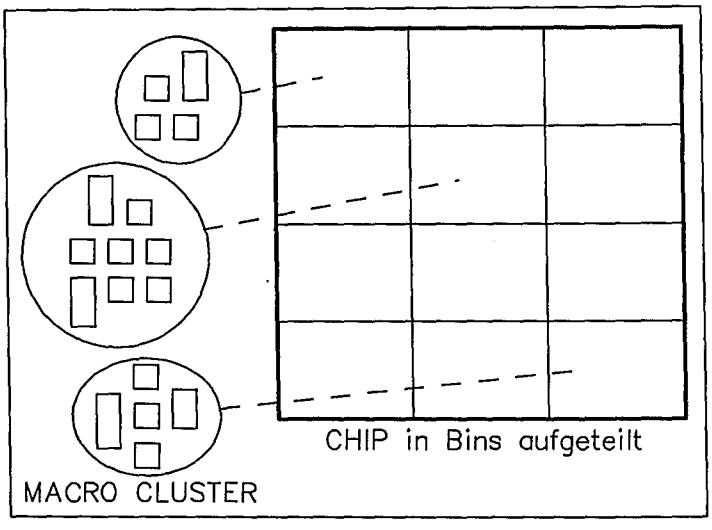


Bild 6.4: Plazierungs-Bins die für globale Plazierung von Clustern (nach GATE-STATION-Handbuch, MENTOR GRAPHICS)

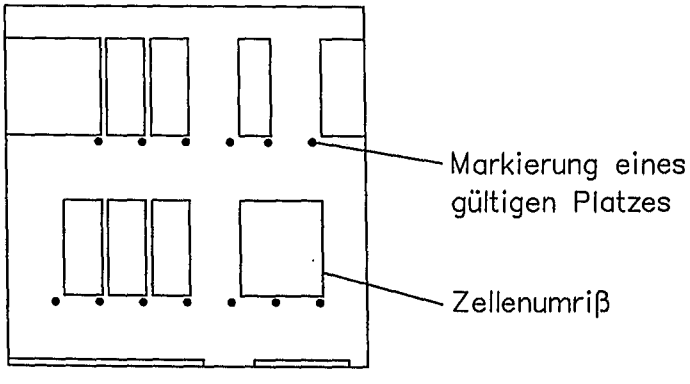


Bild 6.5: Detaillierte Plazierung von Zellen auf gültige Plätze

6.2 Verdrahtung

Nachdem die Zellen plaziert sind, müssen die Verbindungen zwischen den Zellen hergestellt werden. Ein Netz verbindet die Anschlüsse (Pins) der Zellen. Die Verbindung kann in mehreren Ebenen (layers) aus Polysilizium oder mehreren Metallagen geführt werden. Für die Verdrahtung wird die Chipfläche wieder in rechteckige Teilflächen (Bins, coarse grid) aufgeteilt. *Kanäle* sind rechteckige Flächen, mit Anschlüssen an zwei gegenüberliegenden Seiten. Ein *globaler Router* ordnet jedes Netz einer Menge dieser Flächen zu. Dazu werden beim *Coarse Grid Routing* jeder für die Verdrahtung verfügbaren Fläche Rasterpunkte mit einer Kapazität zugeordnet. Diese Kapazität entspricht der maximalen Zahl von Verbindungen, die über diese Fläche geführt werden kann (siehe Bild 6.6). Anschließend muß für jedes Netz (hier die Menge der zu verbindenden Rasterpunkte) die kürzeste Verbindung gefunden werden. Dies kann durch Ermittlung der minimalen Steiner–Bäume erfolgen. Häufig werden jedoch *Maze–Running*–Verfahren verwendet, die auf dem Lee–Algorithmus basieren. Dabei werden alle Rasterpukte um den Ausgangspunkt eines Netzes mit ihrer Entfernung vom Ausgangspunkt numeriert (siehe Bild 6.7a). Ein minimal langer Pfad kann durch Rückwärtsverfolgung vom Ziel zum Ausgangspunkt entlang von Punkten absteigender Numerierung ausgewählt werden (siehe Bild 6.7b). Die Kapazität der dabei benützten Rasterpunkte wird um Eins erniedrigt. Rasterpunkte mit der Kapazität 0 werden als Hindernisse angesehen.

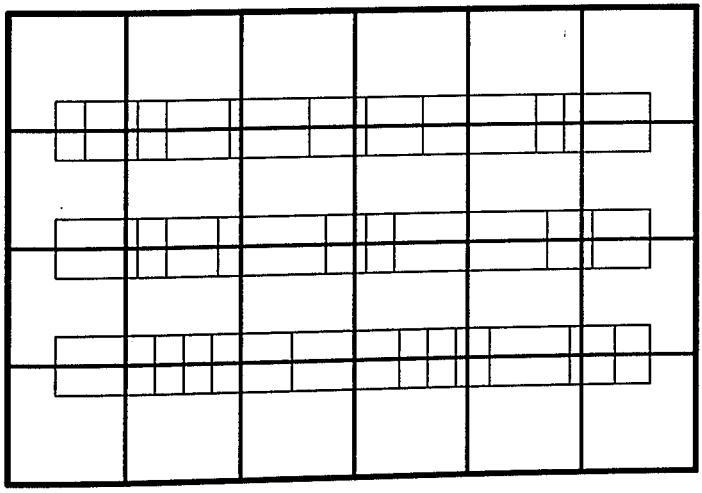


Bild 6.6: In Routing Bins aufgeteilte Chip–Oberfläche

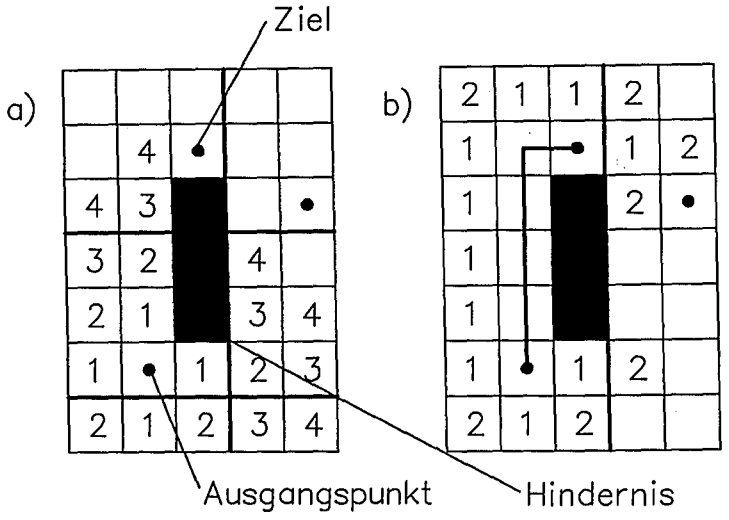


Bild 6.7: Maze/Lee-Routingverfahren

Bei dieser Vorgehensweise werden die Netze sequentiell verlegt. Das Ergebnis hängt dabei stark von der gewählten Reihenfolge ab, in der die Netze verlegt werden. Für die Festlegung einer optimalen Reihenfolge gibt es allerdings keine allgemeingültigen Kriterien. Deshalb wird eine Vorauswahl durch Berücksichtigung *kritischer Netze* getroffen. Kritische Netze sollen möglichst kurz sein, um kleine Verzögerungszeiten zu verursachen. Sie werden deshalb zuerst geroutet, da zu Anfang die wenigsten Hindernisse existieren und der Maze-Algorithmus immer den kürzesten Weg findet. Weitere Ansätze sind das Simulated Annealing nach demselben Verfahren wie beim globalen Plazieren.

Die globale Verdrahtung legt fest, welche Netze durch welche Verdrahtungsflächen (Kanäle) zu führen sind. Die lokale Verdrahtung erfolgt nach unterschiedlichen Verfahren durch den Kanalrouter. Beim Kanalverdrahtungsverfahren wird gewöhnlich von zwei Verdrahtungsebenen ausgegangen. Der Kanal hat dazu horizontale Verdrahtungsleitungen (Spuren) und vertikale Verdrahtungsmöglichkeiten (Spalten). Zusätzlich sind Durchkontaktierungen (Vias) von einer Verdrahtungsebene zur anderen möglich. Bild 6.8 zeigt einen Ausschnitt aus einem Gate-Array. Zu sehen sind die Zellenreihen mit den noch nicht vergebenen Basiszellen (CPWR) und den zwischen den Zellenreihen liegenden Verdrahtungskanälen. Die Zeilen- und Spaltennummern kennzeichnen das Verdrahtungsraster. Plaziert und verdrahtet sind in diesem Bild ein D-Flipflop (DF08), ein Inverter (IN01) und ein Nand-Gatter (NA02). Die kleinen Rechtecke an den Eckpunkten der Verbindungsnetze sind Vias.

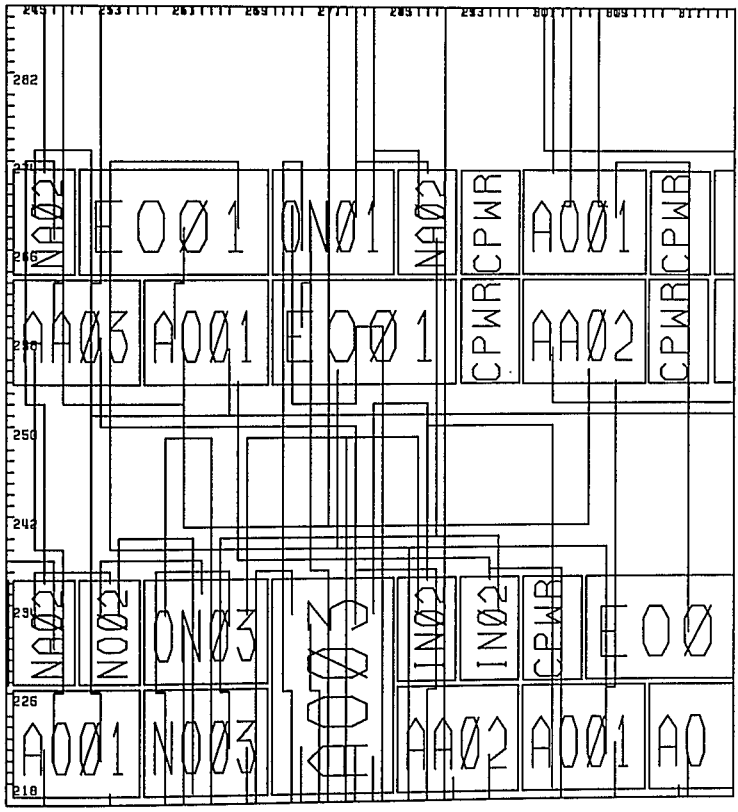


Bild 6.8: Ausschnitt aus einem Gate–Array mit Zellenreihen (CPWR) und Verdrahtungskanälen

Beim *left–edge–Algorithmus* wird ein Netz in einem Kanal über maximal eine Spur geführt, die eine entsprechende Zahl von Spaltenanschlüssen erhält. Beim *dog–leg–Verfahren* kann ein Netz auf mehreren Spuren geführt werden. Die Teilspuren werden durch Spalten verbunden (dog–legs, siehe Bild 6.9). Beim *Greedy*–Routing wird der Kanal von links nach rechts durchlaufen und beim Durchlauf die notwendigen Verbindungen gemäß den folgenden Regeln hergestellt:

- Die Anschlüsse werden von oben bzw. unten in den Kanal eingebracht. Dazu wird die nächste freie Spur oder eine Spur benützt, die das Netz schon enthält, abhängig davon, welche Spur näher ist.
- Es werden *dog–legs* eingeführt, um die in verschiedenen Spuren geführten Teilnetze zusammenzufassen.
- Netze werden möglichst weit nach oben bzw. unten gebracht.

- Alle Netze, die noch weiter rechts liegende Anschlüsse haben oder mehr als eine Spur benutzen, werden zur nächsten Spalte nach rechts weitergeführt. Mit diesem, in groben Zügen skizzierten Verfahren, wird die dichteste Kanalpackung erreicht.

Weitere Routing-Optimierungen sind möglich, wenn äquivalente Makrozellen-Pins ausgetauscht werden (Pin-Swapping). Beispielsweise sind die beiden Eingänge eines 2-Input-NAND vertauschbar, ohne daß die Funktion beeinträchtigt wird. Zusätzlich sind bei vielen Makrozellen mehrere äquivalente Pins verfügbar. Durch Restrukturierung eines Netzes und Tausch äquivalenter Pins kann sich eine Routing-Vereinfachung ergeben. Schließlich ist es möglich Makrozellen zu spiegeln.

Sollte es letztlich nicht möglich sein, Pins an Netze anzuschließen (blocked pins), so werden die Netze in der Nachbarschaft des blockierten Pins wieder aufgetrennt und zunächst das blockierte Netz gelegt. Anschließend wird versucht, die aufgetrennten Netze erneut zu routen (rip up and retry). Falls sich daraus eine Verbesserung des Verdrahtungsergebnisses ergibt, wird diese Änderung beibehalten. Bild 6.10 zeigt einen verdrahteten Kanalausschnitt aus einem Gate-Array.

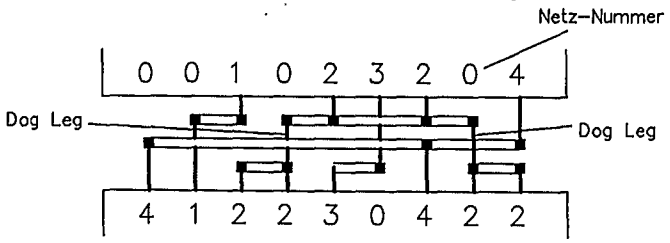


Bild 6.9: Kanalverdrahtung nach dem dog-leg-Verfahren (schematisch)

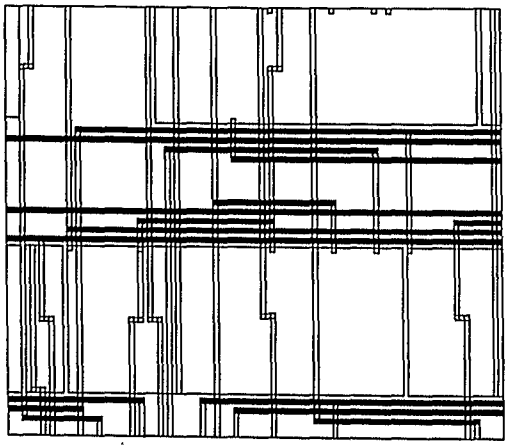


Bild 6.10: Verdrahtungsausschnitt aus einem Gate-Array

6.3 Vorbereitung des Gate–Array Layouts auf Mentor Graphics Workstations

Voraussetzung für die Erstellung eines Layouts ist ein konsistenter, mit NETED und SYMED erstellter Schaltplan, aus dem mit den Software–Werkzeugen EXPAND und EXTRACT sowohl die *Netzliste* mit den Design–Properties (EXPAND_DESIGN) und den Layout–Properties (EXPAND_COMP) gewonnen und in den Files *design.erel* bzw. *comp.erel* abgelegt werden. Zum Schaltplan der obersten Hierarchieebene muß ein *ROOT–Symbol* für den Chip vorhanden sein. Den Pins des ROOT–Symbols müssen die Input–/Output–Makrozellen zugeordnet sein. Die Symbole der I/O–Makrozellen tragen wesentliche Layout–Properties.

Nach fehlerfreier Expandierung der Schaltung muß die erzeugte Datenbasis in eine für die Layoutwerkzeuge kompatible Form konvertiert werden. Dazu dient das Software–Werkzeug LOGIC_ENTRY, das sowohl auf die mit EXPAND/EXTRACT generierte Information, als auch auf die Layoutbibliothek zugreift und dabei eine Ausnutzungsstatistik des Basis–Arrays erstellt. Nach diesem Schritt kann das Layout automatisch mit GATEPLACE und GATEROUTE bzw. interaktiv mit GATEGRAPH durchgeführt werden. Vorher ist es zweckmäßig, Vorplazierungen vorzunehmen und ggf. kritische Netze zu kennzeichnen (Vergabe von Netzprioritäten).

Die wichtigste Vorplazierung ist die Plazierung der I/O–Makrozellen durch Zuordnung der I/O–Signale zu Pins des gewünschten Chip–Gehäuses. Diese Vorplazierung kann bequem mit dem Software–Werkzeug PKG_DEF unter Angabe des Basis–Arrays und des Gehäusetyps erfolgen. PKG_DEF benützt NETED als Benutzerschnittstelle (siehe Bild 6.11). Im Zeichenfenster erscheint das Chipgehäuse. An jedem Pin ist die Verwendungsmöglichkeit als Signal– oder Stromversorgungspin angegeben. Zur Zuordnung eines Signals wird der Signalname im Fenster *Buffer Preplacement* "angeklickt", der Cursor über den Text IONET am Gehäuse–Pin gesetzt und RETURN eingegeben. Abgeschlossen wird durch "Anklicken" von $_EXIT. Die mit PKG_DEF vorplazierten Makrozellen werden von den Layoutwerkzeugen nicht mehr verschoben (protected placement).

Weitere Vorplazierungen vor Beginn der eigentlichen Layoutphase sind bei der Schaltplanerstellung durch Vergabe der SEED–Property unter Angabe eines gültigen Makrozellenplatzes möglich. Falls die Vorplazierung als "protected" gekennzeichnet ist, wird sie durch die Layoutwerkzeuge nicht verändert, andernfalls wird sie lediglich als Ausgangsplazierung verwendet und ggf. abgeändert.

Schließlich können einzelne Netze mit dem NETED–Menü *MakeNetCritical* als kritische Netze markiert werden. Dazu wird Netzen ein Netzprioritätswert zugeordnet. Netze höherer Priorität werden bei automatischer Verdrahtung vor Netzen niedrigerer Priorität geroutet. Netzprioritäten sollten allerdings möglichst selten vergeben werden, da sie das Verdrahtungsergebnis für andere Netze negativ beeinflussen können.

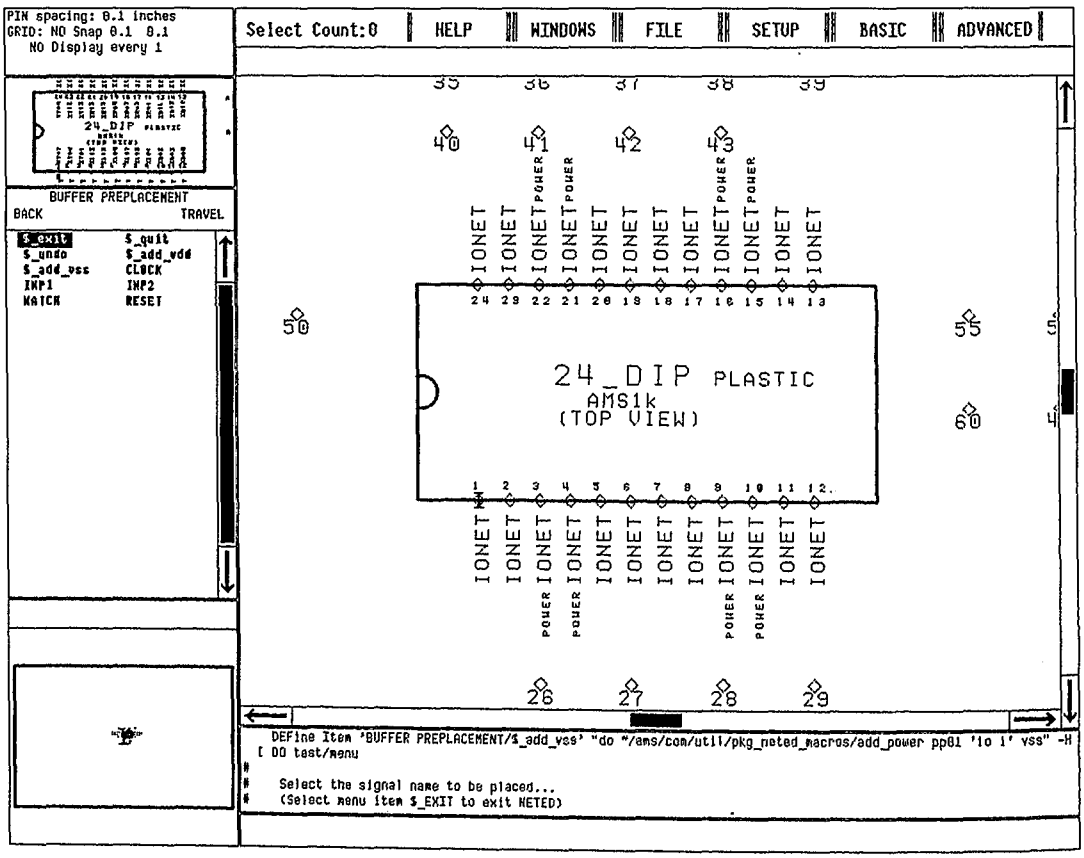


Bild 6.11: Plazierung der I/O-Makrozellen durch Zuordnung der Signalnamen zu den Gehäuse-Pins mit NETED

6.4 Durchführung und Backannotation des Layouts auf Mentor Graphics Workstations

Nach diesen Vorbereitungen kann das Layout erstellt werden. Die Plazierung ist automatisch mit GATEPLACE oder manuell mit GATEGRAPH, aber auch abwechselnd manuell/automatisch möglich. Beispielsweise können mit GATEGRAPH weitere interaktive "protected" Vorplazierungen vorgenommen und die restlichen Plazierungen durch GATEPLACE automatisch vervollständigt werden. Schließlich können automatische Plazierungen interaktiv mit GATEGRAPH abgeändert werden.

In ähnlicher Weise kann die Verdrahtung automatisch mit GATEROUTE oder interaktiv mit GATEGRAPH erfolgen. Auch hier kann beliebig zwischen interaktiven und automatischen Schritten gewechselt werden. So können kritische Netze manuell vorverlegt werden und die Restverdrahtung mit POSTROUTE kann automatisch erfolgen. Automatische Verdrahtungen von GATEROUTE können nachträglich interaktiv mit GATEGRAPH modifiziert werden. Während GATEROUTE nach einer 100 %–Verdrahtung immer ein konsistentes Ergebnis produziert, muß eine Konsistenzprüfung nach einem manuellen Schritt mit GATEGRAPH explizit durch das Werkzeug POSTROUTE vorgenommen werden.

Nach erfolgtem Layout kann dies mit dem Werkzeug GATEPLOT ausgegeben werden. Da GATEGRAPH ein anderes Datenformat benützt, muß beim Wechsel von den automatischen Layoutwerkzeugen zu GATEGRAPH eine Formatkonversion mit PREGRAPH vorgenommen werden.

Vor der Durchführung des Layouts muß sich die *Timingsimulation* der Schaltung auf geschätzte Verzögerungszeiten auf der Basis eines **Technology–Files** (siehe Bild 6.12) abstützen. Dazu wird das Werkzeug ADD_DELAY benützt, das aus der Netzliste die Zahl der Netzverbindungen entnimmt und damit aus dem Technologiefile die geschätzte Kapazität des Netzes auffindet und in der Simulator–Datenbasis die Signallaufzeiten korrigiert.

Nach erfolgtem Layout sind erstmals genaue Verdrahtungslängen und die Zahl der Vias eines Netzes bekannt. Deshalb kann die obige Schätzung durch eine "genaue" Berechnung ersetzt werden. Auch hierzu wird das Werkzeug ADD_DELAY verwendet, allerdings mit der Aufrufoption –BA (backannotation = nachträgliche Anmerkung). Zu diesem Zweck werden die entsprechenden Basisdaten einem weiteren Technologiefile entnommen, das die Kapazität eines Netzes pro Mikron und die Kapazität eines Vias enthält (siehe Bild 6.13).

Im Anschluß an die Backannotation muß die Schaltung einer Nachsimulation unterzogen werden, um die Funktionsfähigkeit unter den geänderten Zeitverhältnissen zu verifizieren.

```
#-------------------------------------------------------------
#  Add_Delay Technologie specification file
#-------------------------------------------------------------
#
# Technology constants and defaults...
PFS_PER_LOAD=0.15;
KRISE_DEFAULT=1.0;
KFALL_DEFAULT=1.0;
#
# Temperature equation based on units of KELVIN...
# F(t) = 0.0000047(t**2) + 0.002565(t) - 0.18494
TEMPERATURE_COEF=-0.18494 0;
TEMPERATURE_COEF=0.002565 1;
TEMPERATURE_COEF=0.0000047 2;
#
# Define the valid temperature range (in CENTIGRADE)...
TEMPERATURE_MAX=135.0;
TEMPERATURE_MIN=-55.0;
#
# Voltage equation based on units of (1/VOLTS)...
# F(v) = 22.8(1/v**2) - 3.557(1/v) + 0.8202
VOLTAGE_COEF=0.8202 0;
VOLTAGE_COEF=-3.557 1;
VOLTAGE_COEF=22.8 2;
#
# Define the valid voltage range (in VOLTS)...
VOLTAGE_MAX=5.5;
VOLTAGE_MIN=4.5;
#
# Delay modification due to processing conditions...
PROCESS_BEST=0.56;
PROCESS_NOMINAL=1.00;
PROCESS_WORST=1.75;
#
# Estimated net capacitance based on connections (in PFS)...
# (A net with 3 connections will have a capacitance of 0.207 pfs)
EXPECTED_NET_CAP=0.058 0;
#
# Estimated net resistance based on connections...
# (A net with 3 connections will have a Resistance of 0.207)
EXPECTED_NET_RES=0.058 1;
EXPECTED_NET_RES=0.115 2;
EXPECTED_NET_RES=0.207 3;
EXPECTED_NET_RES=0.252 4;
EXPECTED_NET_RES=0.370 5;
#
```

Bild 6.12: Technologie–File zur Abschätzung der Signallaufzeiten mit ADD_DELAY (Ausschnitt)

```
Signal net wire data:
 Height & width in microns; capacit. in pfs / linear micron

                          Horizontal          Vertical
Wire type  Plane  Height  Capacitance  Width  Capacitance
-------------------------------------------------------------
    1        1     2.00    0.0000220   2.00   0.0000220
    1        2     2.00    0.0000180   2.00   0.0000180

Signal net via data:
 Height & width in microns; capacitance in pfs / via

Wire type  Plane  Height  Width  Capacitance  Cell name
-------------------------------------------------------
    1        1     3.00    3.00   0.0001700    VIACELL
    1        2     3.00    3.00
```

Bild 6.13: Angaben zur Berechnung der Kapazitäten für die Netze und Vias mit ADD_DELAY... -BA

6.5 Übergabe der Datenbasis an den Halbleiterhersteller

Aus den jetzt vorliegenden Daten wird die Beschreibung des Chip–Layouts in einem vom Hersteller vorgeschriebenen Datenformat generiert und zusammen mit den Testvektoren übergeben. Das Datenformat ist häufig herstellerspezifisch, doch setzt sich mehr und mehr das EDIF–Format (*E*lectronic *D*esign *I*nterchange *F*ormat) durch. Der Hersteller fertigt aus diesen Daten unmittelbar die Verdrahtungsmasken und die Chipmuster.

Literatur zu Kapitel 6

/1/ W. Rosenstiel, R. Camposano: Entwurf hochintegrierter MOS-Schaltungen. Springer-Verlag; Berlin, Heidelberg, New York, 1989

/2/ P. Ammon: Gate-Arrays. Heidelberg: Hüthig Verlag, 1986

/3/ W. Kern: Anwendungsspezifische Integrierte Schaltungen. Heidelberg: Hüthig Verlag, 1986

/4/ H. Fuchs: Selected Reprints on VLSI Technologies and Computer Graphics. IEEE Computer Society Press, ISBN 0-818600491-3

Fragen zum 6. Kapitel

F6.1 Was versteht man unter "symbolischem Layout" ?

F6.2 Diskutieren Sie Optimierungskriterien für Plazierung und Floorplanning; welche Abschätzungen sind üblich?

F6.3 Gegeben sei die Schaltung nach Bild 3.39; führen Sie für diese Schaltung eine Cluster–Analyse für zwei Bins durch.

F6.4 Das Verhalten eines Autoplacers kann durch eine Annealingtemperatur beeinflußt werden. Was können Sie mit diesem Parameter steuern? Wie erreichen Sie eine gute Plazierung?

F6.5 Erläutern Sie den Unterschied zwischen globaler und detaillierter Plazierung bei Channeled Gate–Arrays?

F6.6 Was versteht man beim "Coarse Grid Routing" unter "Rasterpunktkapazität" ?

F6.7 Erläutern Sie warum das Maze–Running–Verfahren immer den kürzesten Weg für ein Netz findet. Ist der gefundene Weg eindeutig?

F6.8 Wie werden Hindernisse durch das Maze–Running–Verfahren erkannt?

F6.9 Was bedeutet im Zusammenhang mit Chip–Layout "Rip–Up and Retry"?

F6.10 Nennen Sie Möglichkeiten zur Optimierung von Routingergebnissen.

F6.11 Wozu werden Netzprioritäten eingesetzt?

F6.12 Was ist der Unterschied zwischen einem "Seed–Placement" und einem "Protected–Placement"

7. Übungsprogramm

In überschaubaren Beispielen soll für einen Anfänger der Kenntnisstand für den Entwurf integrierter Schaltungen und die Anwendung der Entwurfswerkzeuge hierfür in praktischen Übungen vertieft werden. Die gewählten Beispiele sind bewußt einfach gehalten, damit aber hinsichtlich der Komplexität und der technischen Möglichkeiten nicht typisch für den Entwurf integrierter Schaltungen. In ausgewählten Übungsbeispielen wird in die Methodik des Schaltungsentwurfs unter Anwendung von geeigneten Entwurfswerkzeugen eingeführt. Im wesentlichen geht es um die Vermittlung der Funktionalität. Es ist nicht beabsichtigt Operatorwissen zur optimalen Anwendung der Entwurfswerkzeuge zu trainieren.

Die gewählten Beispiele sind als Anregung für einen praktischen Übungsbetrieb gedacht. Das Lernziel ist dabei das Kennenlernen der wichtigsten Entwurfswerkzeuge zur Aufbereitung eines Entwurfs bis zur simulierten und verifizierten Netzliste und das Üben einer systematischen Vorgehensweise. Für die praktische Durchführung sind im Anhang Bedienungshinweise angegeben. Die Bedienungshinweise gelten für Entwurfswerkzeuge von Firma Mentor-Graphics; sie sind so allgemein wie möglich gehalten, um die einzelnen Bedienungsschritte funktionell auch auf andere CAE-Workstations übertragen zu können. Übung 1 dient einem ersten Kennenlernen der Entwurfswerkzeuge ohne Schaltungshierarchie. Die Übungen 2 bis 5 enthalten die Grundschritte für einen systematischen Entwurf und für die Verifikation einer Logikfunktion. In Übung 6 wird eine Analogschaltung simuliert. In Übung 7 schließlich ist im Rahmen einer kleinen Projektaufgabe ein Übungsentwurf zu realisieren. Die durchzuführenden Einzelschritte sind in Anhang G für ein Beispiel zusammengestellt. Im Rahmen der Projektaufgabe sind auch Fragen der Testbarkeit zu berücksichtigen. Das Design ist einschließlich Layout aufzubereiten, so daß eine vollständige Datenbasis mit allen Entwurfsmerkmalen einem Halbleiterhersteller zur Produktion übergeben werden kann.

Übung 1 :	*Einführung in die CAE-Workstation. Kennenlernen der wichtigsten Entwurfswerkzeuge zur Schaltungsverifikation.*
Vorbereitung:	Erste Schritte von Anhang A,B und C durcharbeiten;
Aufgabenstellung:	Realisierung eines 8-Bit-Schieberegisters nach Bild 7.1; Erstellung eines Simulations-Setup-Files; Erstellung eines Force-Files;
Durchführung:	Grafische Schaltplaneingabe des Schieberegisters und anschließende Verifikation durch Logiksimulation.
Bericht:	1. Erläuterung der Aufgabenstellung; 2. Erklärung der Testvektoren (Force-File); 3. Dokumentieren Sie den Sitzungsablauf; 4. Dokumentieren und erläutern Sie: Schaltplan, Set-Up-File, Force-File 5. Diskussion der Ergebnisse der Logiksimulation.

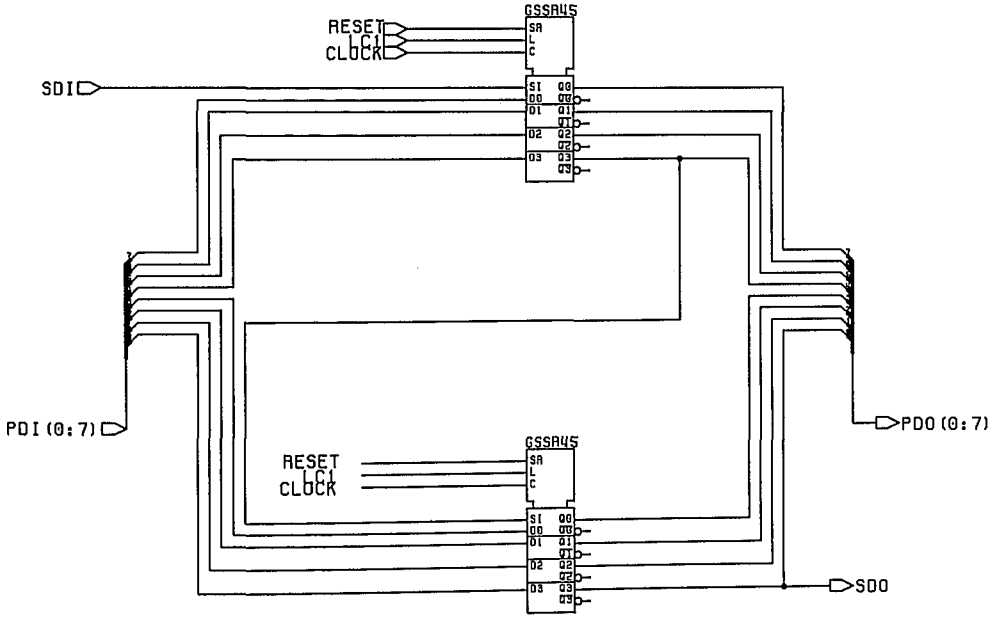


Bild 7.1: 8-Bit-Schieberegister realsiert mit zwei 4-Bit-Schieberegister-Makrozellen

Übung 2 : *Logischer Entwurf einer Ablaufsteuerung; Entwicklung der Ablaufsteuerung als PLD.*

Vorbereitung: Einarbeitung in den systematischen Entwurf von Schaltwerken; Erstellung eines Zustandsübergangsdiagramms.

Aufgabenstellung: Die Multiplizierschaltung nach Bild 2.23 soll dahingehend modifiziert werden, daß sowohl das OR-Register mit dem 1.Operanden als auch das AC-MQ-Register mit dem 2.Operanden direkt ladbar ist (diese Modifikation ist in Bild 7.2b skizziert). Es vereinfacht sich hierfür die Ablaufsteuerung. Entwickeln Sie eine Ablaufsteuerung für die modifizierte Multiplizierschaltung.

Durchführung: Entwickeln Sie die Ablaufsteuerung als PLD.

Bericht:
1. Logiksynthese der Ablaufsteuerung in allen Einzelheiten;
2. Dokumentieren Sie den Sitzungsablauf;
3. Dokumentieren und erläutern Sie das Ergebnis der Logiksynthese.

Übung 3 : *Aufbau einer hierarchischen Schaltungsstruktur (Register–Transferebene).*

Vorbereitung: Anhang B durcharbeiten.

Aufgabenstellung: Blockdefinition und Aufbau einer Registerstruktur in der 2. Hierarchiestufe.

Durchführung: Definition eines Funktionsblocks in der ersten Hierarchiestufe; Vervollständigung der Registerstruktur in der 2. Hierarchiestufe um einen Block für die Ablaufsteuerung; Schaltplaneingabe der Ablaufsteuerung in der 3. Hierarchiestufe.

Bericht:

1. Erläuterung der Aufgabenstellung;
2. Dokumentieren Sie den Sitzungsablauf;
3. Ergänzen Sie die Dokumentation durch den hierarchischen Schaltplan.

Übung 4 : *Funktionelle Verifikation einer Schaltungsstruktur.*

Vorbereitung: Anhang C durcharbeiten.

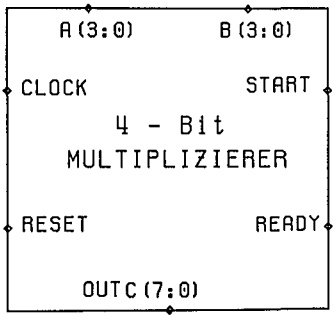
Aufgabenstellung: Erstellung eines Simulations- und MISL–Files für die Gesamtschaltung und funktionelle Schaltungsverifikation.

Durchführung: Logiksimulation der Multiplizierschaltung mit Spike–Analyse und Timing–Analyse.

Bericht:

1. Erläuterung der Aufgabenstellung;
2. Erklärung der Testvektoren (MISL–Files);
3. Dokumentieren Sie den Sitzungsablauf;
4. Ergänzen Sie die Dokumentation durch:
   Simulationsfile, MISL–File und
   dem Ergebnis der Logiksimulation.
5. Diskutieren Sie das Ergebnis der Logiksimulation.

a)



b)

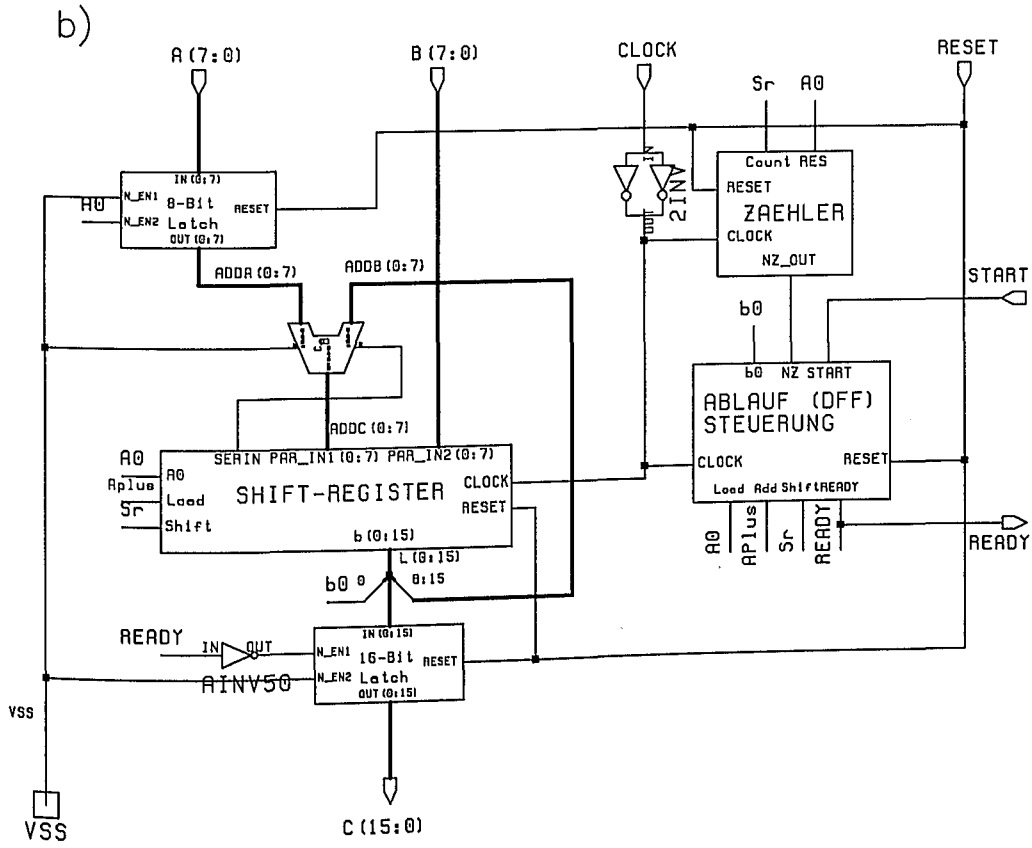


Bild 7.2: Aufbau des Multiplizierers mit direkt ladbaren Operanden; der Ausgangs-Bus C(15:0) ist "gelatcht"

Übung 5 : *Fehlersimulation von Logikschaltungen.*

Vorbereitung: Anhang E durcharbeiten.

Aufgabenstellung: Entwicklung von Teststimuli für eine Logikschaltung (siehe Bild 7.3).

Durchführung: Fehlersimulation und Ermittlung der Fehlerabdeckungsquote; Entwicklung eines Testvektors für 100% Fehlerabdeckung.

Bericht:
1. Erläuterung der Aufgabenstellung;
2. Überlegungen zur Teststimuli-Entwicklung;
3. Dokumentieren Sie den Sitzungsablauf;
4. Dokumentation und Diskussion der Fehlersimulation.

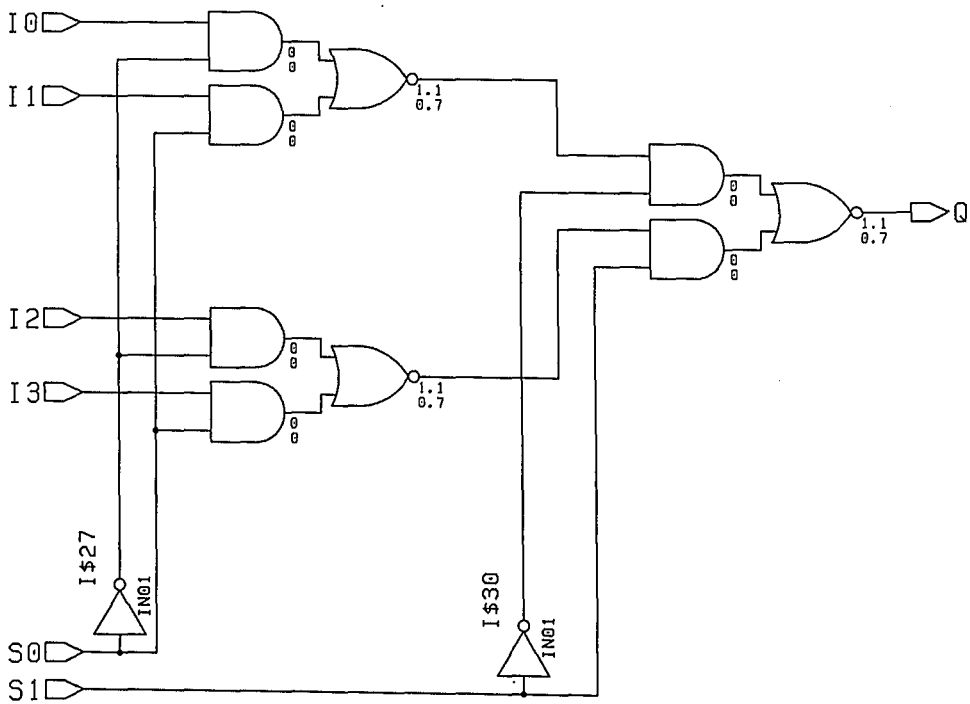


Bild 7.3: Beispiel zur Entwicklung geeigneter Teststimuli; Überprüfung der Teststimuli mit dem Fehlersimulator

Übung 6 : *Circuit–Simulation einer Analogschaltung.*

Vorbereitung: Anhang D durcharbeiten.

Aufgabenstellung: Simulation des Timingverhaltens eines A/D–Wandlers (siehe Bild 7.4).

Durchführung: Grafische Schaltplaneingabe (hierarchisch) und anschließende Simulation mit einem Circuit–Simulator; Bestimmung des Timing–Verhaltens und der Aussteuergrenzen.

Bericht:

1. Erläuterung der Aufgabenstellung und der Schaltung;
2. Dokumentieren Sie den Sitzungsablauf;
3. Dokumentieren und erläutern Sie die Ergebnisse der Circuit–Simulation; DC–Transferverhalten; TRAN–Analyse.

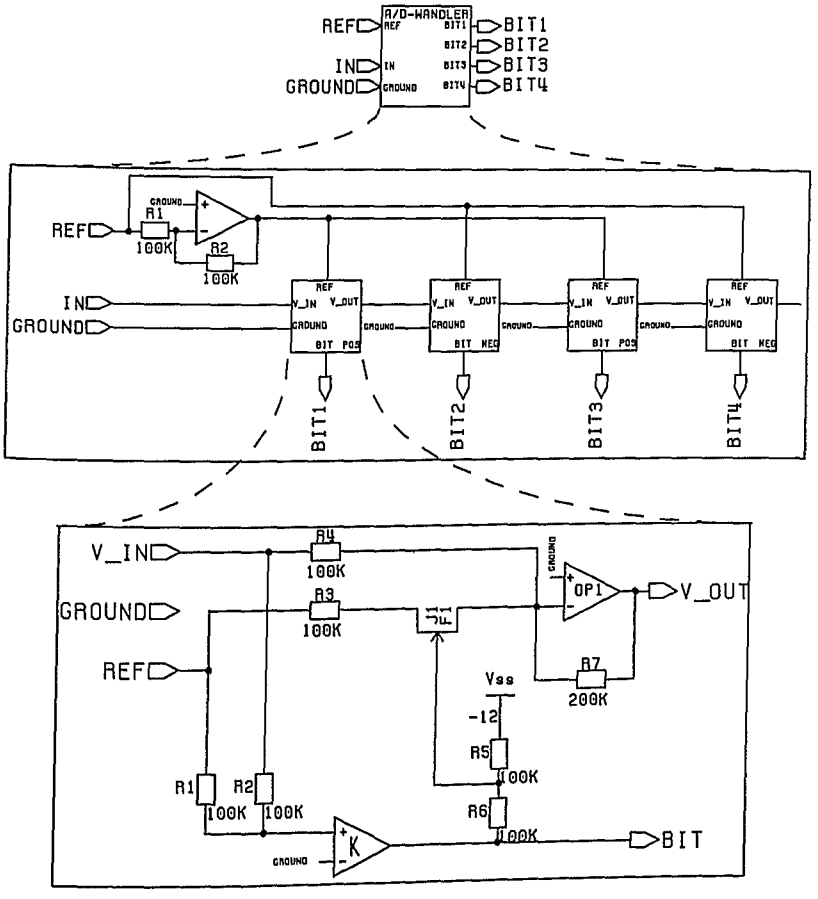


Bild 7.4: Hierachisches Design eines A/D–Wandlers

Übung 7 : *Projektaufgabe: Gate–Array–Design.*

Vorbereitung: Zusätzlich Anhang F und G durcharbeiten.

Aufgabenstellung: Entwickeln Sie einen synchronen Bitstromvergleicher. Bild 7.5 zeigt die Blockfunktion mit den Schnittstellensignalen. Die Bitströme sind taktsynchron. Am Ausgang signalisiert das Signal MATCH durch `1` wenn immer die vollständige Bitfolge `00101` an beiden Bitstromeingängen simultan auftrat.

Durchführung: Spezifizieren Sie den Bitstromvergleicher als Blockfunktion; Entwerfen Sie die Schaltung; Geben Sie die Schaltung hierarchisch gegliedert mit NETED™ an einer CAE–Workstation ein; Verifizieren Sie die Schaltung mit QUICKSIM™; Erstellen Sie ein Chip–Layout; Ermitteln Sie die maximale Taktfrequenz unter worst–case–Bedingungen; Erstellen Sie die Testvektoren und überprüfen Sie Ihre Testvektoren mit QUICKFAULT™; Ändern Sie bei unzureichender Testbarkeit Ihre Schaltung in eine SCAN–TEST–Struktur zur Erhöhung der Testabdeckung.

Bericht:
1. Spezifikation der Aufgabenstellung;
2. Ausführliche Dokumention und Erläuterung aller Einzelschritte;
3. Dokumentation und Diskussion des Entwurfsergebnisses;
4. Diskussion der Testvektoren und der Fehlerabdeckung.

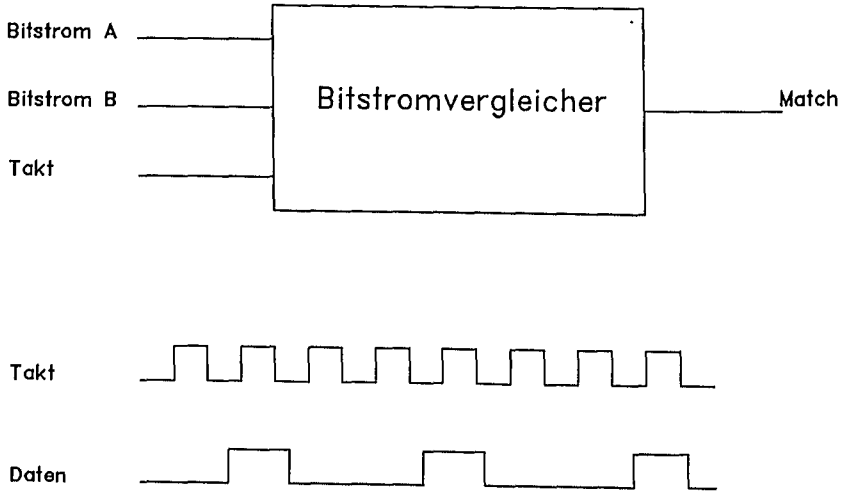


Bild 7.5: Projektaufgabe Bitstromvergleicher

Übung 8 : *Projektaufgabe: Test eines ASIC–Bausteins.*

Vorbereitung: Kurzanleitung über den Tester durcharbeiten.

Aufgabenstellung: Der Baustein von Übung 7 wurde produziert, er soll nunmehr real getestet werden.

Durchführung: Führen Sie einen Hardwaretest des Bausteins durch gemäß Spezifikation.

Bericht:

1. Spezifikation der Testaufgabe;
2. Ausführliche Dokumention und Erläuterung aller Einzelschritte;
3. Dokumentieren und diskutieren Sie das Testergebnis.

Anhang A – Einführung in CAE–Workstations

1. Grundsätzliches zur Rechnerbedienung

1. Schritt: Einloggen

Die Benutzung der CAE–Workstation erfordert eine Autorisierung durch einen USERNAMEN und ein PASSWORT.

Please log in: prakt_x <CR> (Prakt_1..8 je nach Gruppe)

Password: <CR>

Nach erfolgreichem Einloggen befinden Sie sich in einem Directory mit dem Pfadnamen:

/user/prakt_x

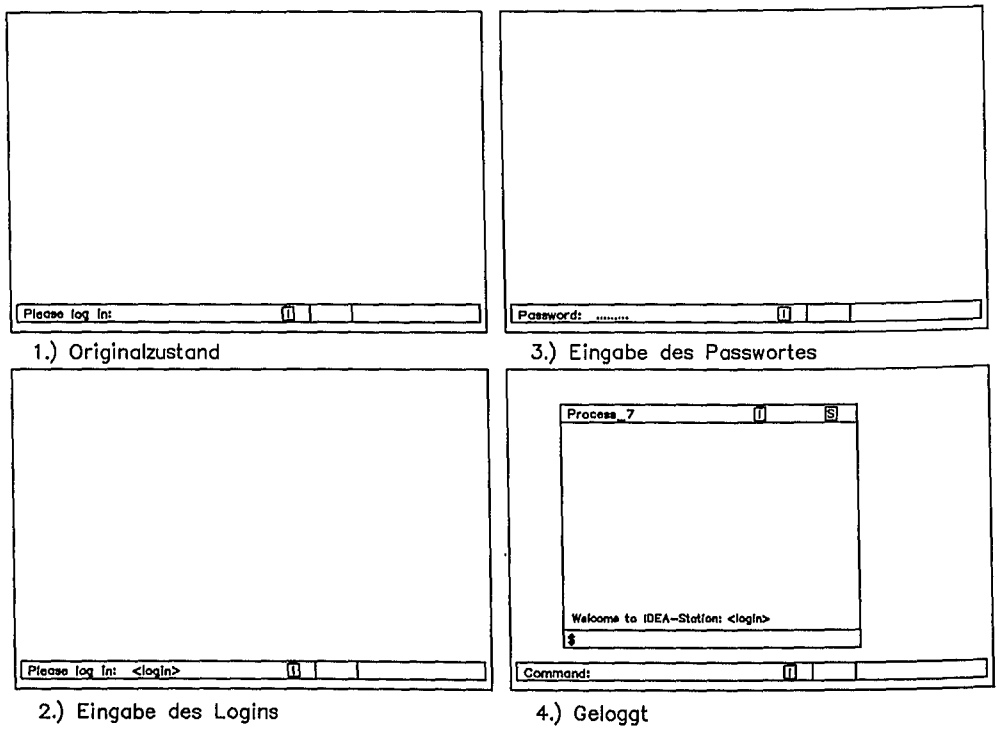


Bild A–1: Das Einloggen an einer CAE–Workstation

2. Schritt: Unterverzeichnis erzeugen bzw. ins Unterverzeichnis wechseln

Nach dem ersten Einloggen (zu Beginn des Praktikums) muß ein Subdirectory erzeugt werden, welches im weiteren Verlauf des Praktikums als Arbeitsdirectory dient.

z.B. : **mkdir** NT7_SS89_G31 (MAKE **DIR**ECTORY)

Anschließend wird durch Verwendung des *Working Directory*–Kommandos **cd** ins Arbeitsdirectory übergewechselt.

z.B. : **cd** NT7_SS89_G31 (CHANGE **DIR**ECTORY)

Bei allen späteren LOGINs ist nach dem Einloggen durch Eingabe des **cd**–Kommandos in dieses Gruppen–Arbeitsdirectory zu wechseln. Mit dem Kommando **pwd** kann der Pfad des aktuellen Directories erfragt werden. Dies sollte vor Beginn jeder Entwicklungsarbeit durchgeführt werden.

Kommandos (z.B. **mkdir, cd, ..,**) oder Aufrufe von Werkzeugen (z.B. neted, symed,..) werden im Process–Fenster nach dem "$"–Zeichen eingegeben (siehe Bild A–1).

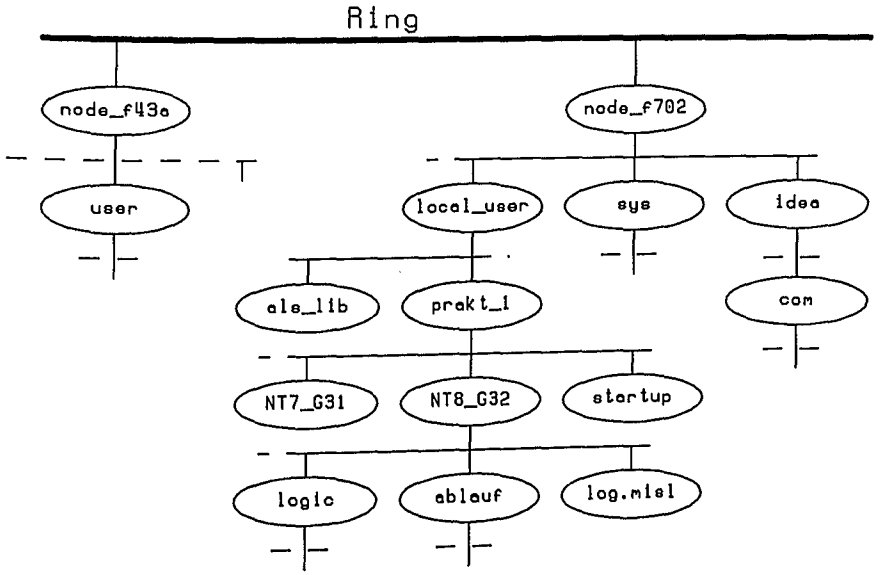


Bild A–2: Netz– und Fileaufbau

3. Schritt: Sitzung beenden

Mit der Maus (od. durch Drücken der Taste CMD) den Cursor in das Command–Window bringen und **LO** <CR> eingeben (**LO** ≡ **LO**GOUT).

2. Bedienung des Editors

Der Editor wird zur Erzeugung von ASCII–Files verwendet. Setup–, MISL– und Simulations–Files werden mit dem Editor erzeugt.

1. Schritt: Aufruf des Editors
Funktionstaste EDIT (Tastenblock r.o.) drücken und Filenamen eingeben.

Edit file: <Filename> <CR>

Es öffnet sich ein Edit–Fenster mit dem Namen <Filename>, welches editiert werden kann (Bild A–3).

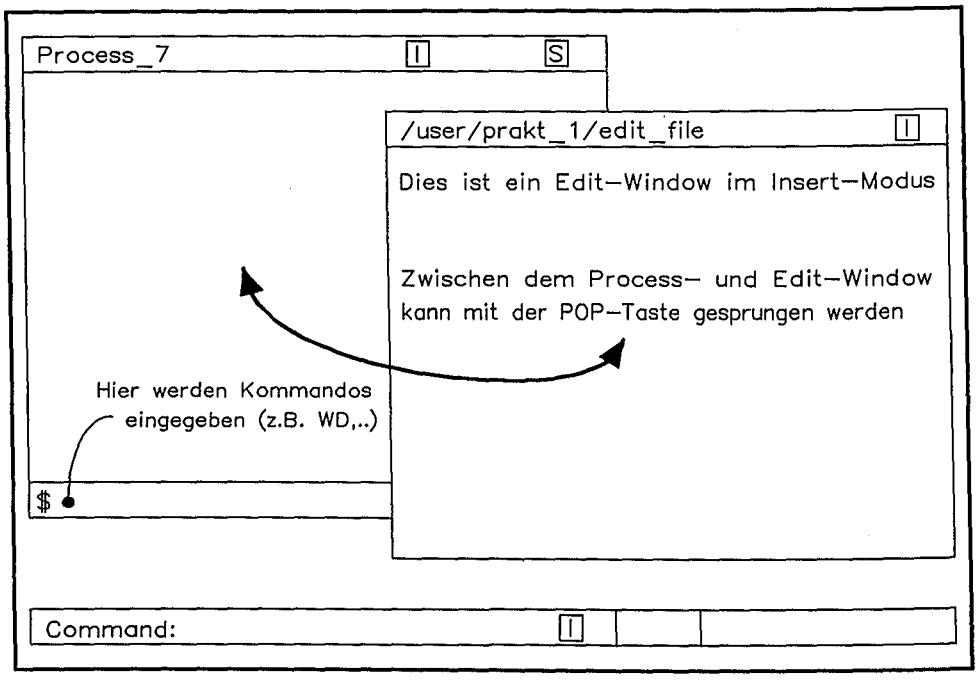


Bild A–3: Prozessfenster am Bildschirm

2. Schritt: Editieren

Es gibt 2 Modi: I für Insert-Modus;
. für Replace-Modus; Umschalten mit Shift&INS

Editiert wird mit den Funktionstasten (Tastenblock links oben):

<CHAR DEL>	Löscht ein Zeichen;
<LINE DEL>	Löscht eine Zeile;
<SHIFT>&<INS>	Umschalten zwischen Editiermodi;
<MARK>	Setzt Markierungspunkt auf die aktuelle Cursor-Position;
<COPY>	Markierten Bereich in Paste-Buffer lesen;
<SHIFT>&<CUT>	Markierten Bereich in Paste-Buffer lesen und löschen;
<PASTE>	Paste-Buffer an Cursor-Postiton auslesen;
<SHIFT>&<UNDO>	Anweisungen (z.B. DEL, PASTE, ..) rückgängig machen.

Textbereich kopieren bzw. löschen:

1. Cursor auf den Anfangspunkt setzen;
2. <MARK>-Taste drücken;
3. Cursor zum Endpunkt bewegen (Bereich markieren);
4. a) <COPY>-Taste drücken (zum Kopieren);
   b) <CUT>-Taste drücken (zum Löschen);
5. <PASTE>-Taste (Text an Cursor-Position schreiben).

3. Schritt: Schließen des Edit-Fensters

Das Edit-Fenster muß nach Abschluß der Editierarbeit durch Drücken der EXIT-Taste (Tastenblock rechts oben) geschlossen werden. Dabei muß sich der Cursor im Edit-Fenster befinden.

3. Wichtige Grundkommandos und Funktionstasten

Grundkommandos :

cd <path>	**C**hange **D**irectory
cd ..	Setze Directory auf nächst höhere Hierarchie
pwd	**P**rint **W**orking **D**irectory
ls	**L**ist Directory
ls –l	**L**ist Directory mit allen Attributen
rm <pathn>	**R**emove File or Directory
mkdir <pathn>	**M**ake **D**irectory
cp <sourcepath> <targetpath>	**C**opy File or Directory
mv <sourcepath> <targetpath>	**M**ove File or Directory

Druckausgabe:

prf <pathn>	Ausdruck auf Nadeldrucker (nur ASCII)
iprf <pathn>	Ausdruck auf Laserdrucker (nur ASCII)

Textbereich einer Protokollausgabe in Paste–Buffer kopieren und ausdrucken:

1. Cursor auf den Anfangspunkt setzen
2. <MARK>–Taste drücken
3. Cursor zum Endpunkt bewegen (Bereich markieren)
4. <COPY>–Taste drücken (zum Kopieren)
5. Cursor in Shell–Command–Zeile und prf eingeben ($ prf <CR>)
6. <PASTE>–Taste drücken (Text an Cursor–Position schreiben)
7. <CTRL>&<B> (Cursor zum Textende)
8. <CTRL>&<Z> (EOF)

Prozeßfenster öffnen/schließen:

Neues Shell eröffnen:	*Shell schließen:*
<SHIFT>&<SHELL>	<CTRL>&<Z> <EXIT>

Maus–Bedienung :

linke Maustaste	Mit dieser Maustaste kann die Fenstergröße verändert werden.
mittlere Maustaste	≡ **cd** <Directory der Cursorposition>
rechte Maustaste	<READ> <File der Cursorposition>

Andere Tasten :

<POP> (Tastenblock u.r.) zwischen den einzelnen Windows blättern.

Anhang B– Schaltplaneingabe mit NETED und SYMED

1. Grundsätzliches zu NETED

NETED™ (Netzwerk–Editor: Mentor Graphics) ist ein Entwurfswerkzeug zur grafischen Schaltplaneingabe mit der Möglichkeit der Bildung von Schaltungshierarchien. Bild B–1 zeigt den Bildschirmaufbau nach Aufruf des Werkzeuges mit den verschiedenen Fenstern.

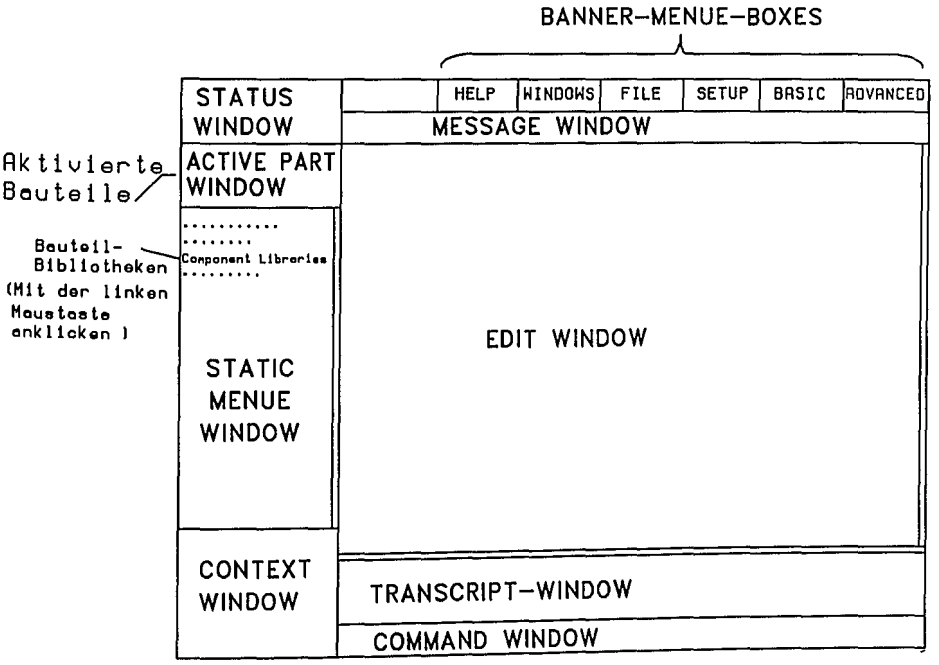


Bild B–1: NETED–Windows

Es gibt drei Möglichkeiten Befehle einzugeben :

1. Durch Pop–Up–Menus*) (Drücken der mittleren Maustaste).
2. Mit Hilfe von Funktionstasten (z.B. <Fx>; <SHIFT>&<Fx>; <CTRL>&<Fx>).
3. Durch Eingabe von Befehlen in der Kommandozeile.

*) Welches Menue erscheint, ist von der Cursor–Position abhängig (z.B. Cursor im Edit–Window -> das Edit–Menue).

Ein Schaltplan besteht im wesentlichen aus drei Grundelementen (Bild B–2):

1. Instances: Ein Instance ist ein Symbol (z.B. Symbol für Widerstand, NAND–Gatter,...) mit einem Handle–Namen (z.B. I$xxx = interne Bezeichnung des Bauteils).

2. Nets: Nets sind Verbindungsnetze zwischen den Symbolen, sie bestehen aus Segmenten und Eck–/Endpunkten (Vertex); Handle–Name für Vertex: V$xxx (= interne Bezeichnung).

3.Pins: Anschlußpunkte an Symbolen.

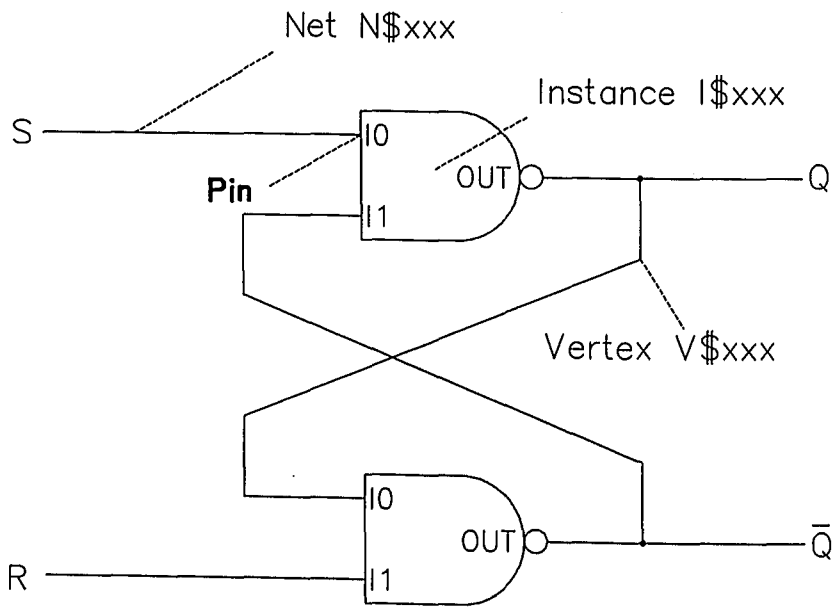


Bild B–2: Zur Bedeutung von Instance, Net, Vertex und Pin

2. Aufruf und Eingabemöglichkeiten im NETED

Verwendete Abkürzungen:

|XX| :	Pop-Up-Window "XX" durch Drücken der mittleren Maustaste im zugehörigen Window aktivieren.
→ "XX" :	Balkencursor im Pop-Up-Window zum Untermenuepunkt "XX" schieben.
↑↓ :	Bedeutet Maus verschieben (z.B. um einen Select-Rahmen aufzuziehen od. ein Bauteil zu plazieren).
& :	Gleichzeitiges Ausführen.
SEL :	Selektieren von Instances, Nets, Pins u.a. mit : F1 ↑↓ oder linke Maustaste drücken ↑↓.
Pfadname:	./<design-name>

1. Schritt: Aufruf von NETED.

$MIKRON_NETED Designfile<CR>

MIKRON_NETED : Batch-File zum Aufruf von NETED mit speziellem STATIC-MENUE-WINDOW.
Designfile : Name des Design welches erzeugt oder geändert werden soll.

Nach der Eingabe obiger Zeile sollte Bild B-1 am Bildschirm zu sehen sein.

2. Schritt: Grundeinstellungen vornehmen (nur bei neuem Design).

Mit dem Pop-Up-Menue (Banner-Menue-Box) SETUP werden die Grundeinstellungen vorgenommen (Bild B-3).

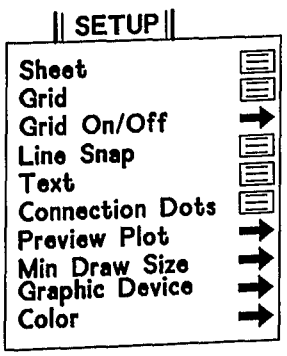


Bild B-3: Menü SETUP

Blattgröße einstellen (Bild B–4):

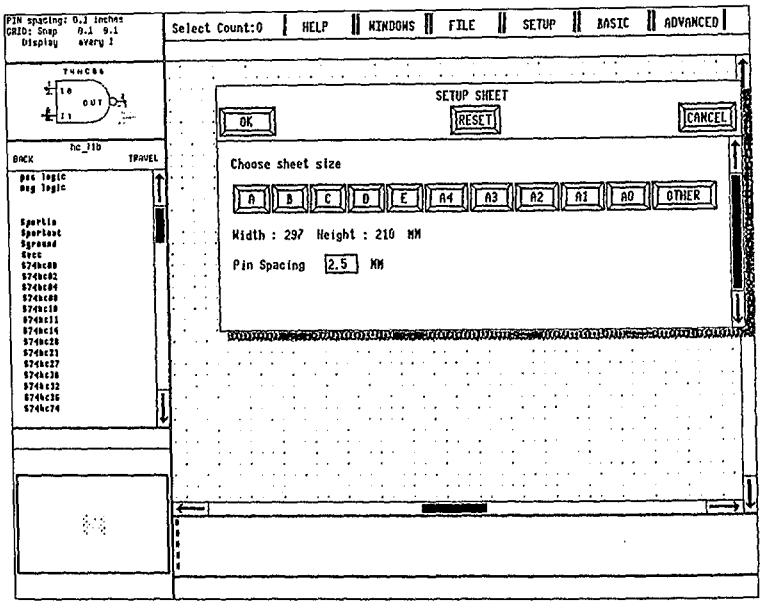


Bild B–4: SETUP -> Sheet (z.B. A4)

Grid (Raster) einstellen (Bild B–5):

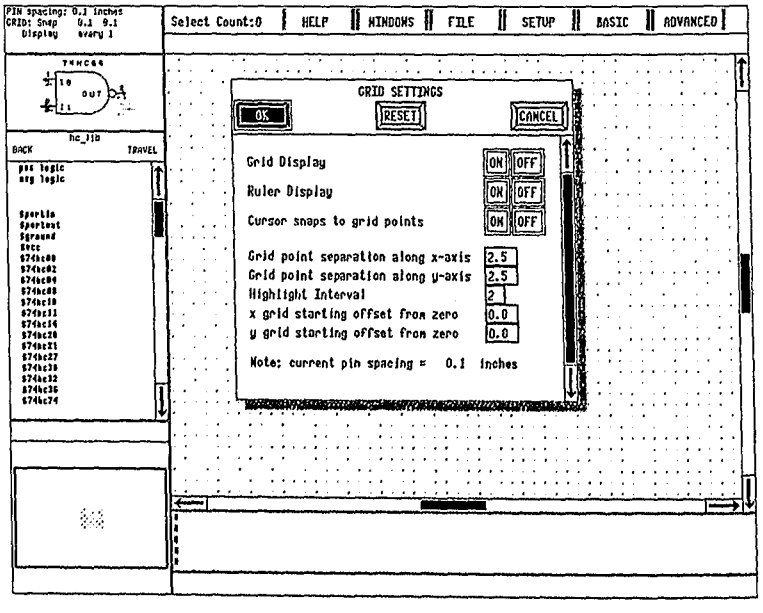


Bild B–5: SETUP -> Grid

Line Snap einstellen (Bild B-6):

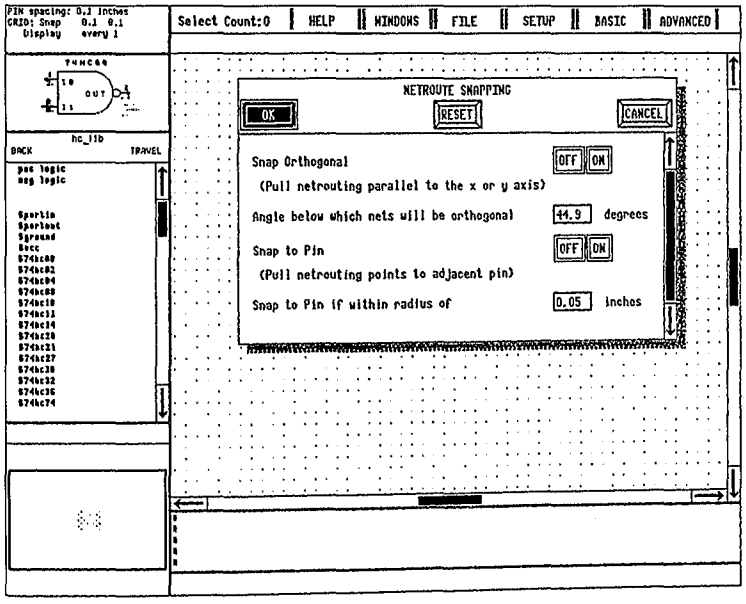


Bild B-6: SETUP -> Line Snap

Text konfigurieren (Bild B-7):

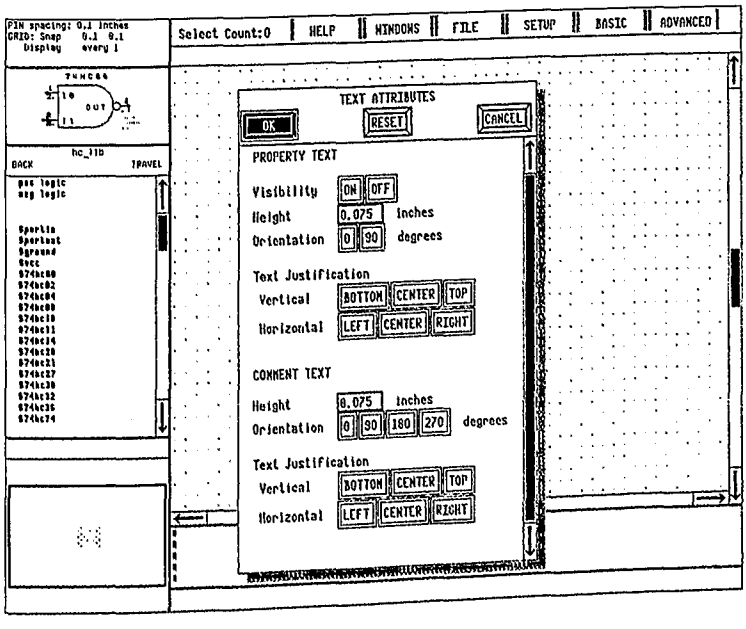


Bild B-7: SETUP -> Text

Connection Dots festlegen (Bild B-8):

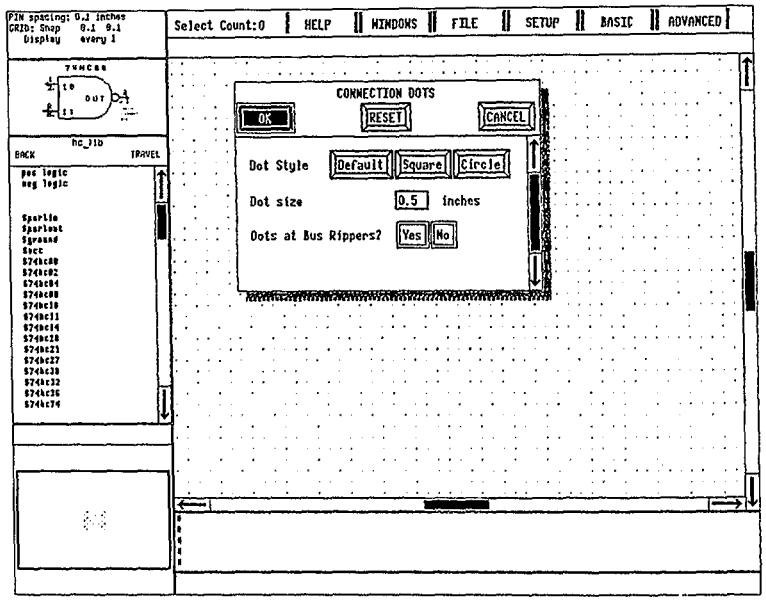


Bild B-8: SETUP -> Connection Dots

3. Schritt: Symbole plazieren.

1. Symbolbibliothek auswählen:

   Um Elemente (z.B. der Mikron_lib) auswählen zu können, müssen Sie im Static-Menue-Window nacheinander folgende Menue-Punkte anklicken (linke Maustaste):

   - Component Library - (Bauteil-Bibliotheken allgemein)
   - Mikron - (Bibliothek von Mikron)
   - Analog/Digital - (analoge oder digitale Elemente)

2. Symbol ins EDIT-Window:

   In der gewählten Bibliothek (STATIC-WINDOW) gewünschtes Bauteil anklikken; es erscheint das Symbol im ACTIVE-PART-WINDOW.

   Es gibt drei Möglichkeiten das Symbol ins EDIT-Window zu übernehmen:

   - |EDIT-WINDOW| → Parts → Place Active Part & ↑↓
   - |ACTIVE-PART-WINDOW| → Place Active Part & ↑↓
   - <F5>& ↑↓

4. Schritt: Pins verbinden und Netze zeichnen.

Es gibt zwei Möglichkeiten Pins zu verbinden:

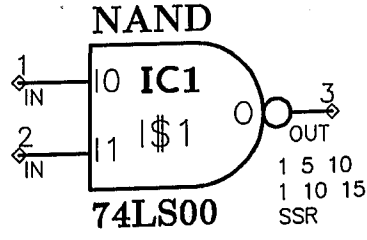
- Mit Menue : |EDIT| → Draw Net → Segment
  1. linke Maustaste anklicken (Segmentanfang) ↑↓
  2. linke Maustaste " (Segmentende)
  3. rechte Maustaste " (= Stift abheben) ↑↓
  4. mittlere Maustaste " (= Draw Net–Modus beenden)
- Mit Funktionstasten : <F3> (Segmentanfang) ↑↓ <F4> (Segmentende)

5. Schritt: Properties verändern oder neu einführen.

Properties sind Attribute zur Festlegung von Kennungen, Eigenschaften und Merkmalen (Bild B–9).

Body–Properties:

Class	*NAND*
Ref	*IC1*
Inst	*I$1*
Comp	*74LS00*



Pin–Properties:

Pin_No.	*3*
Pin	*OUT*
Rise	*1 5 10*
Fall	*1 10 15*
Drive	*SSR*

Bild B–9: Beispiele für Properties am Bauteil bzw. an Pins

1. Ändern :

- SEL(Owner) →|EDIT–WINDOW| → Properties → Modify Properties
  Hiermit erhält man auch den Status der an einem Instance/Pin/Net hinzugefügten Properties.
- oder Cursor auf Property–Text & <SHIFT>&<F7>

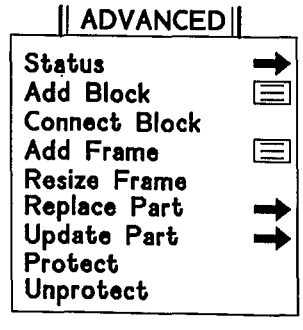
2. Hinzufügen :

- SEL(Owner) →|EDIT–WINDOW| → Properties → Add Properties
  In folgendem Fenster sind Property–Name (z.B. BEZEICHNER) und ein Property–Value einzugeben.

6. Schritt: Block erzeugen.

Bei Erstellung eines hierarchischen Designs wird die Schaltung in einzelne Blöcke zerlegt.

1. *Block erzeugen (Bild B–10):*
   |ADVANCED| → Add Block ↑↓
2. *Netze zeichnen* (4.Schritt)
3. *Blöcke mit Netzen verbinden* (Pins anfügen) :
   SEL(Block) →|ADVANCED| → Connect Block
4. *Blöcke abspeichern :*
   |FILE| → Save
5. Eintauchen in Blöcke :
   SEL(Block) →|FILE| → Open Sheet → Below for Edit



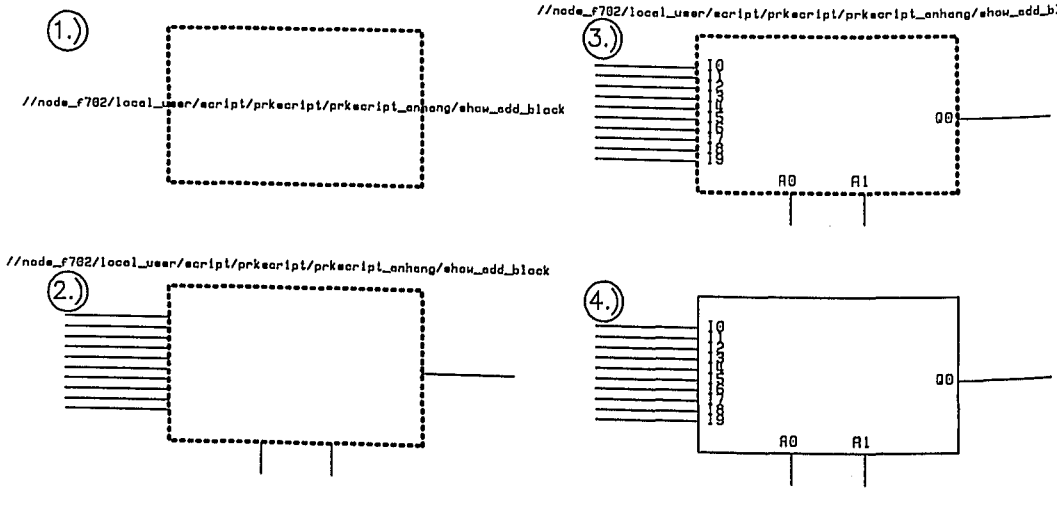
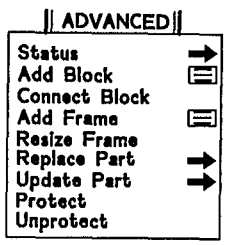


Bild B–10: Erzeugen eines Blockes

7. Schritt: Symbol updaten.

|ADVANCED| → Update Part → Symbol

Dieser Befehl muß nach jeder Änderung eines Blocks angewendet werden. Es erscheint sonst nach der Konsistenzprüfung |BASIC| → Check die Fehlermeldung "OUT OF DATA REFERENCE".



8. Schritt: Netzbündel einführen.

Ein Bus wird durch einen Busbezeichner (z.B. INBus(0:7), OUTBus(1:16)) an einem Netz festgelegt. Zweckmäßigerweise wird der Bus durch eine dickere Strichstärke beim Zeichnen des Busnetzes veranschaulicht (NETSTYLE BOLD: Funktionstaste Shift&F3). Die abgehenden Einzelnetze müssen über Busabgriffe angeschlossen werden. Die Zuordnung der jeweiligen Busleitung erfolgt durch Vergabe der RULE PROPERTY am Busabgriff. Der Busabgriff (Bauteil $RIP) selbst ist ein parametrisiertes Symbol mit dem Laufparameter TYPE. Je nach benötigtem Typ des Busabgriffs muß dem Laufparameter der entsprechende Wert zugewiesen werden. In Bild B-11 ist ein Bus mit Busabgriffen dargestellt.

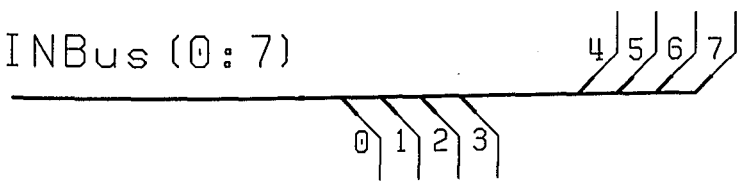


Bild B-11: Busabgriff

Bild B-12 zeigt eine Übersicht des Elementes $RIP mit den verschiedenen Werten für den Laufparameter TYPE. In Bild B-13 ist dargestellt, wie ein derartiges Symbol zu laden ist.

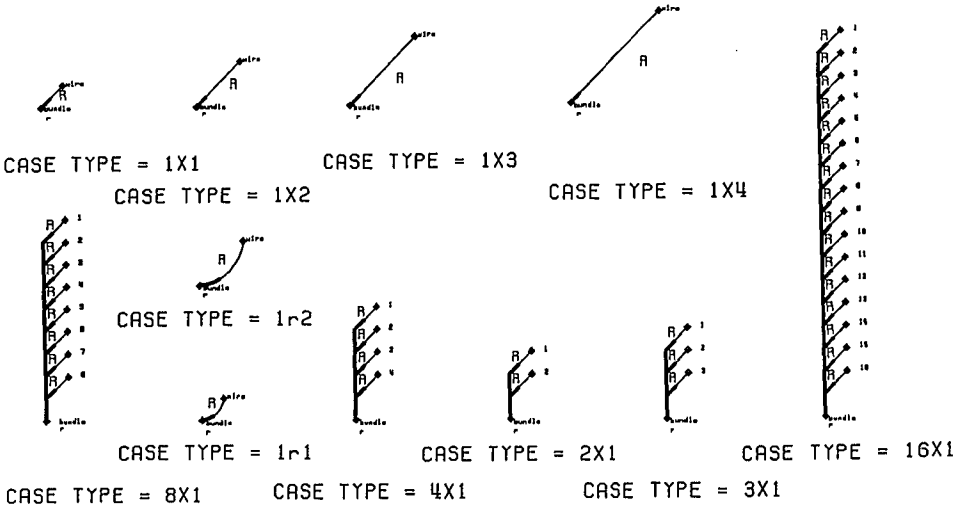


Bild B–12: Busabgriffarten

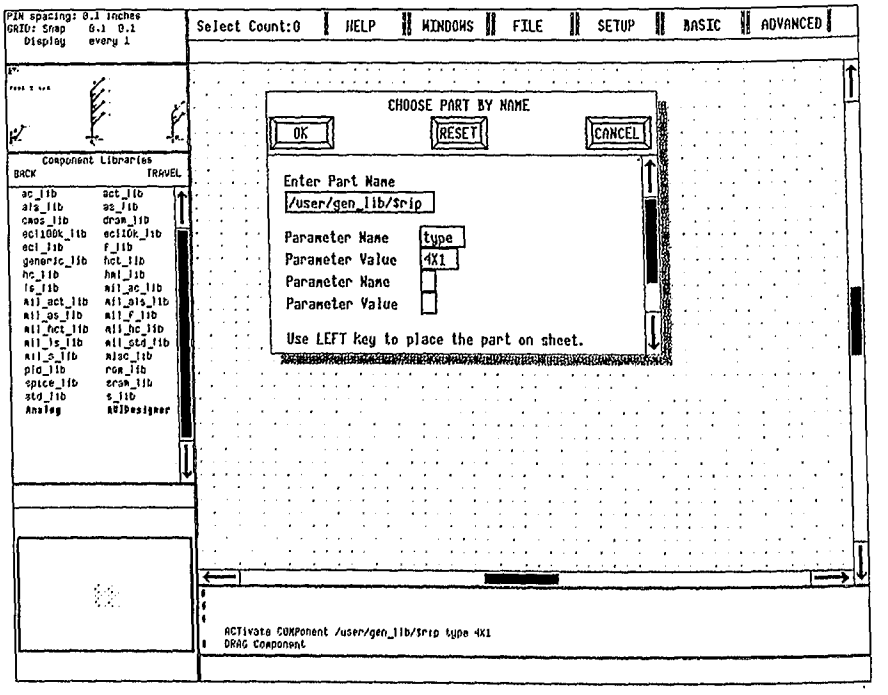


Bild B–13: Laden des Symbols $RIP in das Edit–Window

9. Schritt: Frame-Element erzeugen.

Gruppen von Schaltungsteilen (z.B. Bustreiber) werden zweckmäßigerweise als Frame-Element definiert. Die Zuordnung zu den einzelnen Netzen erfolgt über einen Laufparameter.

- Schaltungsteil erstellen mit Laufparametern
- Rahmen definieren: |ADVANCED| → ADD FRAME

Bild B-14 zeigt ein Frame-Element als bidirektionalen Bustreiber mit Tri-State-Buffern. Über ENABLE wird OUT_INT bzw. IN_INT auf SIG geschaltet.

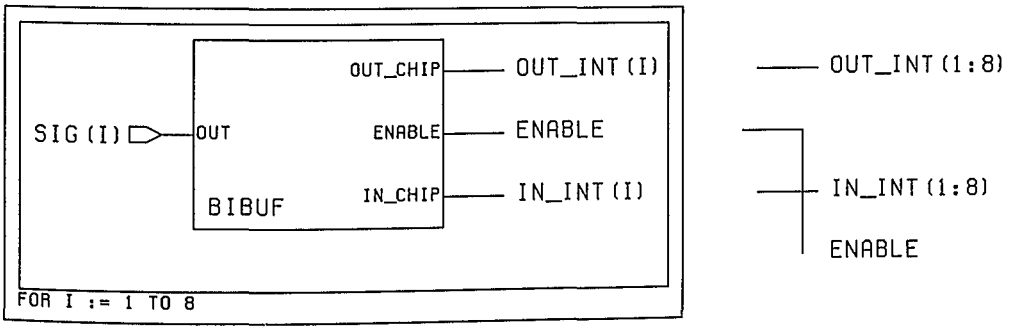


Bild B-14: Frame-Element mit den Anschlussnetzen

10. Schritt: Abspeichern.

- Schaltung überprüfen : |BASIC| → Check
  Untersucht das Design auf Korrektheit und " Vollständigkeit "
- NETED verlassen : |FILE| → Save & Quit

3. Weitere wichtige Funktionen und Menues

Gesamtes Blatt oder Ausschnitt darstellen

VIEW ALL : <Shift>&<F8>
VIEW AREA : <F8>& ↑↓

.
....Maus verschieben

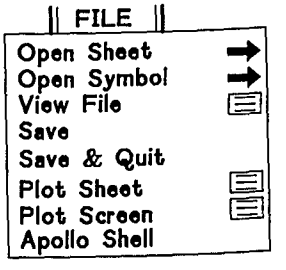
Objekte selektieren oder Select–Mode löschen

SELECT : <F1>& ↑↓
UNSELECT : <F2>

Zeichenblattausgabe

|FILE| → Plot Sheet

FILE–Pop–Up–Menue

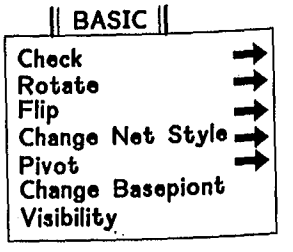


Eine Hierarchieebene tiefer :
SEL(Block)→|FILE|→ Open Sheet → Below for Edit

Eine Hierarchieebene höher :
|FILE| → Open Sheet → Above for Edit

Symbol editieren :
SEL(Symbol)→|FILE|→ Open Symbol → By Selection

BASIC–Pop–Up–Menue



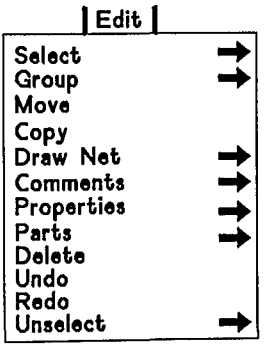
Konsistenzprüfung der Schaltung :
|BASIC| → Check

Bauteile rotieren :
SEL(Bauteil) → |BASIC| → Rotate → –90

Bauteile spiegeln :
SEL(Bauteil) → |BASIC| → Flip → Left–Right

EDIT–Pop–Up–Menue

Select–Element :
|EDIT| → Select → gewünschter Typ ,
SEL(gewünschten Bereich)



Element verschieben :
SEL(Element) → |EDIT| → Move ↑↓ click–li.

Element kopieren :
SEL(Element) → |EDIT| → Copy ↑↓ click–li.

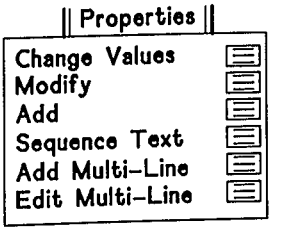
Symbole aufrufen (ins Editfenster holen):
|EDIT| → Parts → By Name

Element löschen :
SEL(Element) → |EDIT| → Delete → Confirm Delete

Property–Menue

|EDIT| → Properties

Property–Werte ändern (mehrere) :
|EDIT| → Properties → Change Values
- Property–Name eingeben (z.B. NET,PIN,..)
- " –Values eingeben (z.B. Netz– od. Pinnamen)
- Cursor auf die jeweiligen Properties setzen und durch klicken der linken Maustaste Property–Wert ändern.



Property–Attribute ändern :
SEL(Property–Owner) → |EDIT| → Properties →
Modify

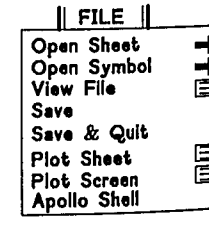
Property–Werte mit Laufvariablen :
|EDIT| → Properties → Sequence Text

4. Symbolerzeugung mit SYMED

SYMED™ (SYMbol EDitor: Mentor Graphics)) ist eine Sonderform von NETED™, speziell für das Erstellen von Symbolen. Der Bildschirmaufbau ist ähnlich der bei der grafischen Schaltplaneingabe.

1. Schritt: Mit Menü den Symboleditor aufrufen.

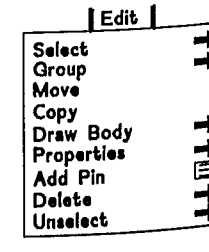
|FILE| → Open Symbol → By Pathname
bei neu zu erstellenden Symbolen und Update;
in das erscheinende Pop-Up-Menü Pfadnamen eintragen
./<design-name>



|FILE| → Open Symbol → By Selection
bei Modifikationen;
das zu ändernde Symbol vorher selektieren.

2. Schritt: Symbolkörper zeichnen.

|Edit| → Draw Body → Rectangle bzw.
|Edit| → Draw Body → Line
zeichnen der Konturen des Symbols; kann bei Modifikationen entfallen.

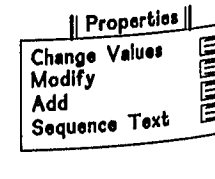


3. Schritt: Pins erzeugen.

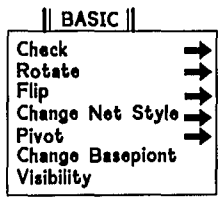
|Edit| → Add Pin
Pin-Bezeichner ins Pop-Up-Menü eingeben und Pin plazieren. Vorgang für alle erforderlichen Pins wiederhohlen.

4. Schritt: Symbol bezeichnen und Text ausrichten.

|Edit| → Properties → Add und
|Edit| → Properties → Modify
Property Name und Property Value in die Pop-Up-Menüs eintragen (editieren), ggf. Attribute ändern.



5. Schritt: Konsistenzprüfung.



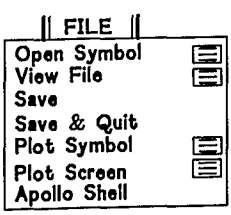
|Basic| → Check
Fehler falls vorhanden beseitigen. Das Pop-Up Fenster mit CLOSE schließen. Das "Checken" verschmilzt alles zu einer Einheit, d.h. einzelne Linien, Rechtecke, Pins usw. bilden nun ein Symbol.

6. Schritt: Definition des Origins.

Definiert den Bezugspunkt (Basepointer), an den das Symbol *in NETED* "aufgehängt" wird. In der Praxis hat es sich als vorteilhaft erwiesen, den Origin eines Symbols an einen Pin in der linken unteren Ecke zu setzen.

Dazu den Cursor an die gewünschte Stelle fahren und über Tastatur den Befehl ORIGIN in die Kommandozeile eingeben.

7. Schritt: Symbol mit Laser ausdrucken.



|File| → Plot Symbol
Scale-Faktor = −1; für Anpassung an DIN A4-Blatt.

8. Schritt: Abspeichern.

|File| → <Save>&<Quit>
Schreibt das Symbol auf die Festplatte.

Anhang C – Logiksimulation mit QUICKSIM

QUICKSIM™ (Mentor Graphics) ist ein interaktiver Logiksimulator. Er erlaubt 12 Simulationszustände und ermöglicht Timingchecks (z.B. Setup- und Holdzeiten, minimale Pulsbreiten).

1. Schritt: Erzeugen der Design-Datenbasis für QUICKSIM.

Zum Zwecke der Simulation mit QUICKSIM wird die Schaltung mit EXPAND expandiert. Dabei wird aus der bisherigen Datenbasis die Information (Properties) extrahiert (EXTRACT), die der Logiksimulator benötigt.

Funktion :

1. "Flachmachen" der Hierarchie;
   einsetzen der funktionalen Blöcke usw. für den Simulator
2. Überprüft:
   - Pin- und Portnamen auf Übereinstimmung
   - Versionsnummern von Bauteilen (neueste Version ?)
   - Existenz von Bauteilen (Pfadnamen)
   - Bus-Bezeichnungen
   - Rule-Properties (Bus Anschlüsse)
3. Legt beim Abspeichern durch Write ein Directory
   <Designname>/design.erel an.

Aufruf :

MIKRON_EXPAND <Designfile> (Bei Verwendung der MIKRON-Bibliothek)

oder AMS_EXPAND_DESIGN <Designfile> ga_2dm (Bei Verwendung der AMS_Bibliothek)

2. Schritt: Vorbereitung der SETUP- und STIMULUS FILES für QUICKSIM.

QUICKSIM benötigt SETUP-Informationen und Eingangssignalfolgen (Stimuli), die über Tastaturkommandos, POPUP-Menüs oder Kommandoprozeduren einzugeben sind. Da die SETUPs und STIMULI öfter benötigt werden (z.B. nach Verbesserung nach Auftreten eines Fehlers) ist es ratsam, Kommandoprozeduren mit den erforderlichen Anweisungen in ein File (siehe EDITOR – Anhang A) zu schreiben und bei Bedarf abzurufen.

- *Beispiel eines SETUP–Files:*

```
# Löschen aller bestehenden Vordefinitionen
#
FORGET BUS -ALL
FORGET SYN -ALL
FORGET FORCE -ALL
FORGET VIEW -ALL
FORGET PROBE -ALL
FORGET TRACE -ALL
#
# Zweckmäßige Synonym–Definitionen
#
SYN SIGN3 /I$1/state(0)
SYN SIGN2 /I$1/state(1)
SYN SIGN1 /I$1/state(2)
#
# Definition eines Signalbündels
#
DEFINE BUS curstate SIGN1 SIGN2 SIGN3 -Combine
#
# Festlegung der darzustellenden Signale
#
TRACE CLOCK RESET
TRACE INP1 INP2 INP3 INP4
TRACE SIGN1 SIGN2 SIGN3
TRACE CURSTATE
#
# Schaltplan anzeigen
#
VIEW SHEET
#
# Signal–Stimulus einlesen entweder :
#DO signal.force
# oder :
READ FORCE MISL signal.misl
#
# Sichern des aktuellen Simulationszustands
#
SAVE STATE T.0 -Replace
```

STIMULI können durch FORCE–Kommandos oder durch Stimulus–Programme in der Programmiersprache MISL vorgegeben werden. MISL–Programme sind zur Vorgabe von Stimulus–Sequenzen zweckmäßiger.

FORCE-Kommandos sind zweckmäßig, wenn in eine Simulation von Hand eingegriffen werden soll. Dies ist immer dann sinnvoll, wenn Netze in der Schaltung nachgesetzt werden sollen.

- *Beispiel eines FORCE-Files :*

```
# Festlegung des CLOCK-Signals als Background-Signal
#
CLOCK PERIOD 200
FORCE CLOCK 1    0.0 -Repeat
FORCE CLOCK 0 100.0 -Repeat
#Festlegung der übrigen Eingangssignale
FORCE RESET 1    0.0
FORCE RESET 0 150.0
FORCE INP1 1    0.0
FORCE INP1 0 250.0
FORCE INP1 1 450.0
FORCE INP2 0    0.0
FORCE INP2 1 850.0
   :      :   :   :
```

- *Beispiel eines MISL-Files :*

```
CIRCUIT testdesign;        /* Programmdeklaration */
INPUT    /inp1             /* Deklaration der Eingangssignale */
         /inp2             /* Hier als TOP-LEVEL-Signale */
           :
         /reset;
OUTPUT   /b0                /* Deklaration der Ausgangssignale */
         /b1                /* Hier als TOP-LEVEL-Signale */
           :                   /* OUTPUT ist für die Fehlersimulation */
         /read;             /* notwendig */
TIMEDEF PERIOD = 50 ns;     /* Dauer eines Simulationszyklus */
DEFAULT = 12.5 ns;           /* Definition einer DEFAULT- */
                             /* Zeitspanne */
BACKGROUND /clock = HI:+2 , LO:+2 .. ; /* Schaltungstakt */

/* Signalprogramme */
/reset = HI ,               /* high zum Zeitpunkt 0 nsec */
        LO:+3;              /* low 3 DEFAULT-Zeitspannen spaeter */
/inp1 = LO ,
        HI:+3,
          :
/xxx  = LO, HI:+8
$                           /* alle Signal-Sequenzen spezifiziert */
                            /* $  begrenzt diese Signalsequenz */
END.                        /* Programm-Ende */
```

3. Schritt: Aufruf von QUICKSIM.

QUICKSIM <Designname>

Der Logiksimulator wird gestartet. Falls die Extraktion für die Simulationsdatenbasis noch nicht erfolgt ist, startet QUICKSIM noch EXTRACT. Nach dem Aufbau der Benutzeroberfläche kann die SETUP-Phase folgen. Bild C-1 zeigt den Bildschirmaufbau von QUICKSIM.

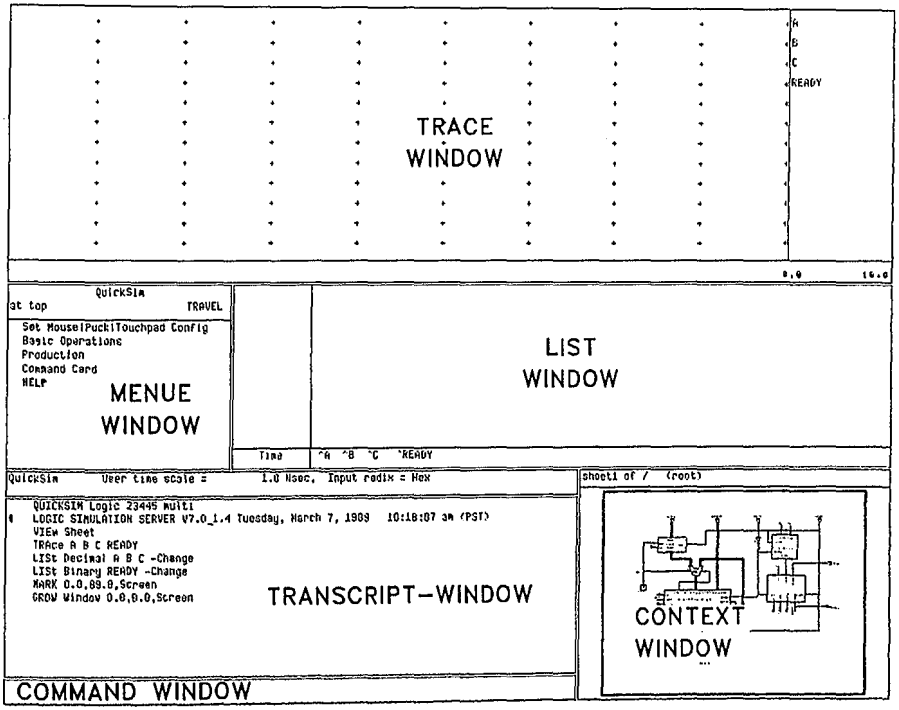


Bild C-1: Bildschirmaufbau bei QUICKSIM

4. Schritt: Setup des Logiksimulators.

DO <Setup-File>

Falls das Setup-File keine Stimuli setzt, sind diese noch manuell zu setzen oder es ist eine Stimulus-Kommandoprozedur aufzurufen. Anschließend ist es zweckmäßig den Simulator-Anfangszustand mit SAVE STATE T.0 -R abzuspeichern. Durch RESTORE STATE T.0 kann der Simulator in den Anfangszustand zurückgesetzt werden, was eine erneute Simulation möglich macht, ohne die Eingangssignale neu zu setzen.

5. Schritt: Simulationszustandsspeicher einschalten.

HISTORY xxxx
(Für die Zeitspanne von xxxx ns speichert QUICKSIM den Logikzustand aller Netze).

Nachtragen von Signalen im Trace-Window:
TRACE signalnamen
REGENERATE TRACE

6. Schritt: Starten der Simulation.

RUN <TIME>

Der Zeitfaktor <TIME> muß bei der Verwendung von Stimuli-Force-Files verwendet werden. <TIME> gibt an wie lange simuliert werden soll.
Bei der Verwendung von MISL-Files legt die letzte Signaländerung die Länge der Simulation fest; es genügt also das Kommando RUN.

In Bild C-2 ist der Ablauf der Logiksimulation dargestellt. Die Logikzustände zu einem bestimmten Zeitpunkt (z.B. State T.3000) können mit SAVE STATE T.3000 abgespeichert werden. Dieser Zustand läßt sich mit RESTORE STATE T.3000 später wieder laden.

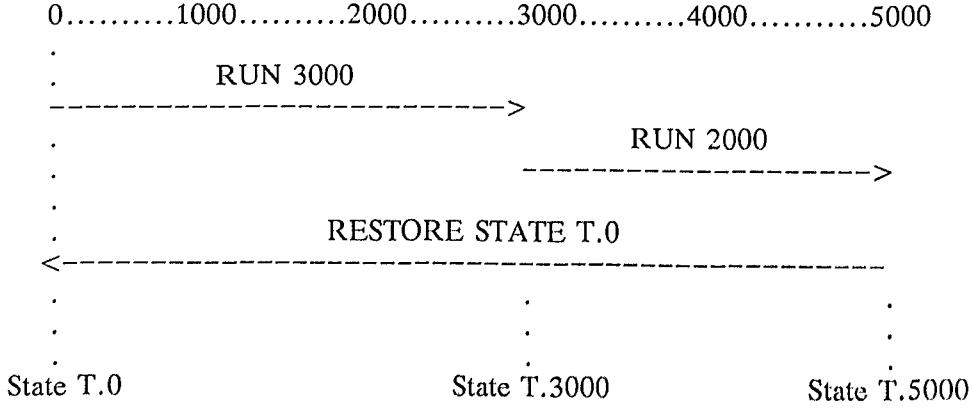


Bild C-2: Zum Ablauf der Logiksimulation

7. Schritt: Verlassen von QUICKSIM.

BYE

Einige QUICKSIM Kommandos:

DO <command-file>	Führt Kommandos im Kommandofile 'command-file' aus (z.B. Setup- oder Force-Kommandos in einem File).
PLOT TRACE	Gibt das Impulsdiagramm auf dem Laserdrukker aus. Vor der PLOT-Anweisung ist <Shift>&<F8> zu drücken um den gesamten Simulationsbereich auf einem Sheet zu erfassen.
PROBE <probe-name>	Dem Netz der Schaltung auf dem der Cursor steht wird der Tastkopf 'probe-name' zugeordnet.
READ FORCE MISL <filename>	Entnimmt Stimuli aus dem MISL-File 'filename' mit Hilfe des Stimulusprogramms.
SAVE STATE <state> -R	Abspeichern des Logikzustands; die Option -R bedeutet Replace-Mode, d.h. Überschreiben eines existierenden Files.
RESTORE STATE <state>	Setzt den QUICKSIM auf den Zustand zurück welcher vorher mit SAVE STATE <state> abgespeichert wurde.

Einige wichtige QUICKSIM-Funktionstasten:

Add Cursor	<F5>
Slide Cursor	<F4>& ↑↓
Select Cursor	<Shift>&<F5>
Eintauchen in das nächst untere Sheet des Blocks, auf dem der Cursor positioniert ist	<CTRL>&<F8>
Zurück in das darüberliegende Sheet	<CTRL>&<F7>
Probe setzen, d.h. einem Netz (selektiert durch den Cursor) einen Namen zur Darstellung des Signalzustands geben.	<Shift>&<F3>

Anhang D – Circuit–Simulation mit MSPICE

MSPICE™ (Mentor Graphics) ist ein Circuitsimulator (Quelle: SPICE) mit einer interaktiven Bedienungsoberfläche u.a. zur Festlegung der Steuerparameter für die Schaltkreissimulation.

1. Schritt: Erzeugung der Daten–Basis für MSPICE.

Zum Zwecke der Simulation mit MSPICE wird die Schaltung mit EXPAND expandiert. Dabei wird aus der bisherigen Datenbasis die Information (Properties) extrahiert (EXTRACT), die der Circuitsimulator benötigt.

Funktion:

1. "Flachmachen" der Hierarchie;
   EXTRAHIEREN aller für SPICE notwendigen Properties.

2. Überprüft:
   - Pin– und Portnamen auf Übereinstimmung
   - Versionsnummern von Bauteilen und (neueste Version ?)
   - Existenz von Bauteilen (Pfadnamen)
   - Bus–Bezeichnungen
   - Rule–Properties (Bus Anschlüsse)

3. Legt beim Abspeichern durch Write ein Directory
   <Designname>/design.erel an.

Aufruf:

$ EXPAND_MSP <Designfile>

2. Schritt: Aufruf von MSPICE.

$MSPICE <Designfile>

MSPICE ermöglicht mit Hilfe von Pop–Up–Menüs die einfache Eingabe von Simulationsart (AC–,DC–,TRANsienten–Analyse) und Eingangssignalen (DC–,SIN– und PULse–Signale).

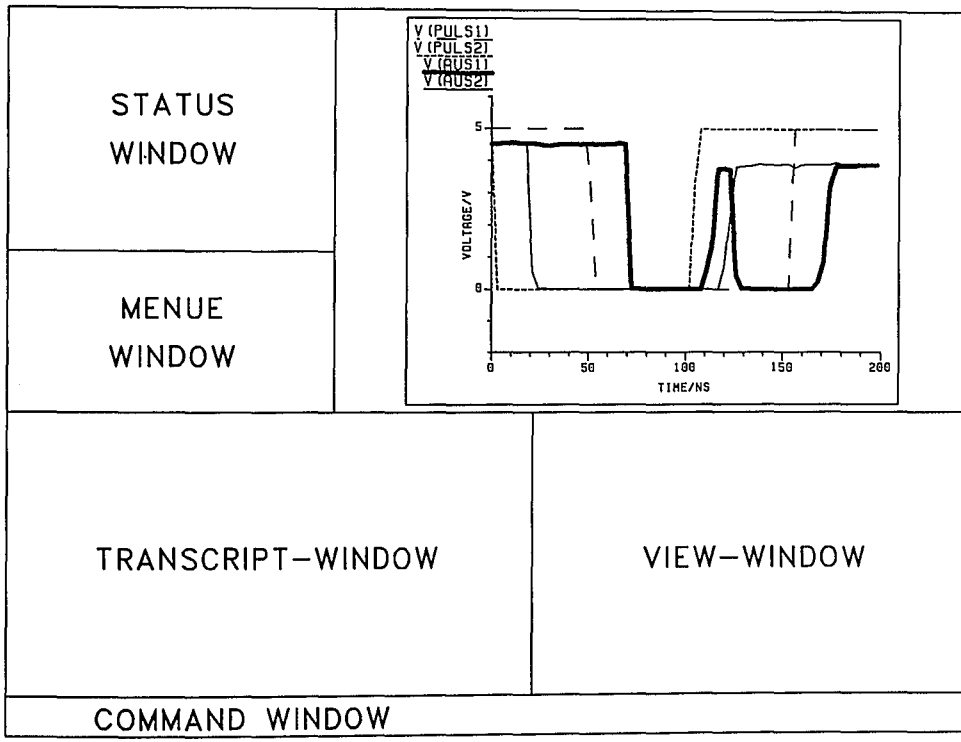


Bild D–1: Bildschirmaufbau von MSPICE

3. Schritt: Festlegung der Eingangssignale (FORCE).

BASIC OPERATIONS → SETUP → FORCE

Force type :	Für DC–Analyse: DC Für AC–Analyse: AC Für TRAN–Analyse: EXP | PUL | PWL | SFFM | SIN
Voltage oder Current ?	Voltage | I
Signal Name :	Namen eintragen
Signalform definieren:	Template ausfüllen

4. Schritt: Festlegung der Simulationsart (ANALYSIS).

BASIC OPERATIONS → SETUP → ANALYSIS

Analysis Mode :	DCOP; DC; AC; TRAN;	
Parameter der Simulationsart festlegen:	Template ausfüllen;	
Models File :	z.B. /user/spice_lib/spice.models	Muß nur einmal eingestellt werden sonst : '' bzw. NO
View Sheet ?	YES	
Spawn Server ?	YES	
Ground :	//GROUND	

Da es sich bei MSPICE "nur" um eine graphische Oberfläche zur einfacheren Benutzung von SPICE und anschaulicheren Darstellung der Simulationsergebnisse handelt, muß ein Mailboxserver, zuständig für den Datentransfer zwischen SPICE und der Benutzeroberfläche, gestartet werden. Gestartet wird der Server durch 'Spawn Server : YES' im Pop–Up–Menü ANALYSIS.

Wird in der Analyseart ein Signalname verlangt (z.B. bei DCOP), so ist vorher dieser Signalname im 3. Schritt zu definieren.

5. Schritt: Start von SPICE.

RUN

eingeben oder Funktionstaste F1 drücken.

6. Schritt: Ergebnisdarstellung mit CHART oder DISPLAY.

- *Mit CHART*

  Über Pop–Up–Menü BASIC OPERATIONS → CHART

  Spannung oder Strom auswählen : z.B. Create Voltage (Current) Graph
  Kurvenfunktion auswählen : z.B. A
  in Kommandozeile : (Enter Signal Name)>...

  Bei Signalnamen sind auch implizite Bezeichnungen möglich.

  Es können 16 Felder dargestellt werden:

```
.....................................
.  A1  .  A2  .  A3  .  A4 .
.....................................
.  B1  .      .      .     .
.....................................
.  C1  .      .      .     .
.....................................
.  D1  .      .      .     .
.....................................
```

  Selektieren eines Charts : <SHIFT>&<F7> (Cursor muß auf gewünschtem Feld stehen)
  oder: SEL CHA A2

  Löschen eines Charts : FORGET CHA A2
  Plotten eines Charts : PLOT 0 A2
  Plotten aller Charts : PLOT 0

- *Mit DISPLAY*

  Über Pop–Up–Menü BASIC OPERATIONS → DISPLAY

  → Trace Voltage — Signalnamen eingeben (Darstellung der Spannungen)
  → Trace Current — Signalnamen eingeben (Darstellung der Ströme)
  → Monitor Voltage — ALL (Darstellung der Spannungen im Sheet)

- *Implizite Signalnamen*

  Netze oder Stromzweige können auch implizit zur Darstellung in CHART festgelegt werden. Als Signalname dient auch der Pinbezeichner eines Symbols, z.B.:

  RName/pos
  RName/neg
  CName/pos
  CName/neg
  ...
  QName/e
  QName/b
  QName/c
  MName/d
  MNAme/g
  MName/s
  ...
  SName/N1
  SName/N2

7. Schritt: Abschluß.

<CTRL>&<Y> oder Eingeben von BYE

Anhang E – Fehlersimulation mit QUICKFAULT

Bei der Entwicklung Integrierter Schaltungen (z.B. ASIC) nimmt die Fehlersimulation eine wichtige Rolle ein. Zum Testen der Schaltungen stehen eine begrenzte Anzahl von Ein- und Ausgangspins zur Verfügung. Der Entwickler versucht nun durch geeignete Wahl von Testvektoren und/-oder durch Anbringung zusätzlicher Testpins eine möglichst hohe Testbarkeit der Schaltung zu erzielen.

Als Werkzeug stehen hierfür dem Entwickler Fehlersimulatoren zur Verfügung, welche "Fabrikationsfehler" nachbilden. Hierzu werden interne Netze auf HIGH- oder LOW gelegt und die Schaltung simuliert.

QUICKFAULT™ (Mentor Graphics) ist ein interaktiver Simulator, welcher als Fehler- oder Logiksimulator einsetzbar ist. Er bestimmt, wie effektiv Testvektoren herstellungsbedingte Schaltkreisfehler erkennen können. Dazu simuliert QUICKFAULT das Verhalten der fehlerfreien Schaltung. Anschließend fügt der Simulator einen Fehler in die Schaltung ein (möglicher realer Fehlerfall) und vergleicht das Simulationsergebnis mit dem Simulationsergebnis der fehlerfreien Simulation. Weichen beide Signalfolgen voneinander ab, so gilt der eingefügte Fehler als durch den Vektorsatz erkannt (*detected fault*). Stimmen die Ausgangssignalfolgen überein, so gilt der eingefügte Fehler als vom Testvektorsatz nicht erkannt (*undetected fault*). Je nach "Fehler-Pegel" spricht man von "stuck-at-1" und "stuck-at-0"-Fehlern. Schließlich gibt es Fälle, bei denen die Simulation des fehlerbehafteten Schaltkreises einen undefinierten Logikpegel ermittelt. Der Fehlersimulator kann also nicht feststellen ob die Ausgangssignalfolge des fehlerbehafteten Chips in der Realität von der Vergleichssignalfolge abweicht oder nicht. Solche Fehler werden als durch den Testvektor möglicherweise feststellbare Fehler (*possibly detected fault*) klassifiziert.

1. Schritt: Vorbereitung der Design-Datenbasis für QUICKFAULT.

Siehe Anhang C: 1.Schritt (analog der Datenbasisgenerierung für QUICKSIM)

2. Schritt: Aufruf von QUICKFAULT.

$ QUICKFAULT <Designname>

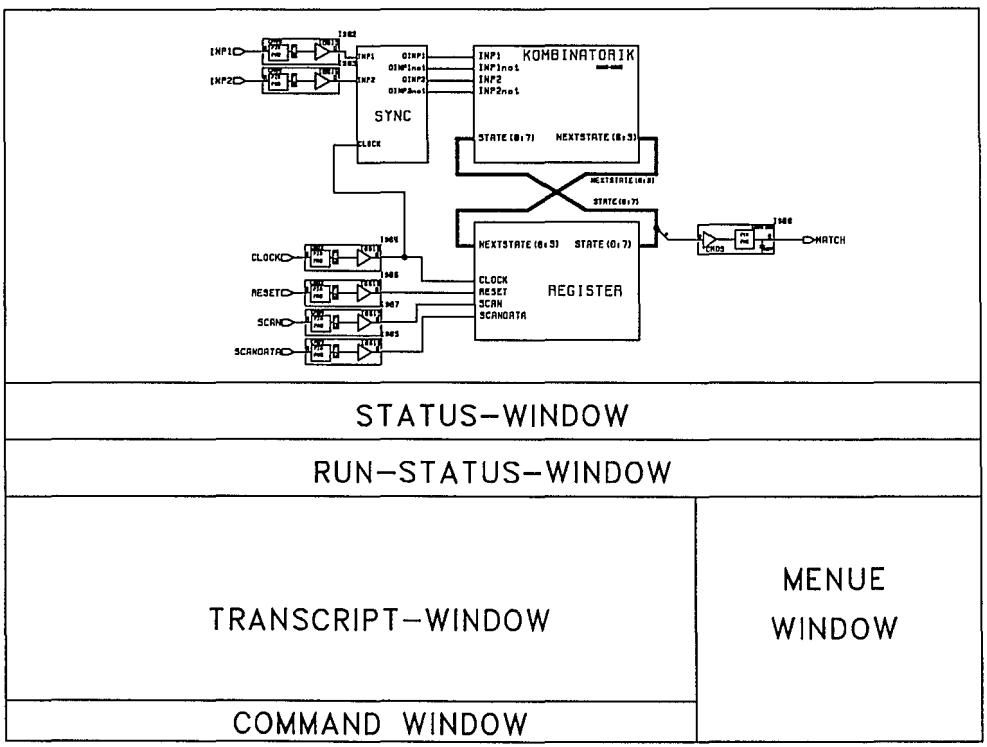


Bild E-1: Bildschirmaufbau von QUICKFAULT

3. Schritt: Aufruf der Stimuli und Simulationsstart.

QuickFault> READ FORCe Misl <Misl-Filename> oder
DO <Force-Filename>

anschließend:

QuickFault> RUN

Das RUN-Kommando startet die Simulation beim Zeitpunkt 0, wenn das Stimuli-File fehlerfrei eingelesen wurde.

4. Schritt: Darstellen der nicht erkannten Fehler (*undetected fault*).

QuickFault> VIEw FAUlts Undetected oder Funktionstaste F7 drücken;
vorher muß mit VIEw SHEET der Schaltplan dargestellt
werden.

Im VIEW-Window werden nach obiger Anweisung Fehler-Fahnen angebracht. Eine Fahne über einem Netz oder in der oberen Hälfte eines Instances kennzeichnet einen "stuck-at-1"-Fehler, eine Fahne unter einem Netz oder in der unteren Hälfte eines Instances kennzeichnet einen "stuck-at-0"-Fehler.

5. Schritt: Ausblenden von Fehlern.

Wichtig ist für den Entwickler die Testbarkeit seines Designs. Die angezeigten Fehler in den Bibliothekszellen (z.B. Flipflops) können im allgemeinen nicht durch schaltungstechnische Maßnahmen eliminiert werden.

Um Fehler von Funktionsblöcken oder Makrozellen auszublenden sind folgende Schritte notwendig:

1. Eintauchen in die Makrozelle
   - Cursor auf Bauteil setzen
   - <CTRL>&<F8> drücken (VIEW DOWN)
   - Darstellen der nicht erkannten Fehler (siehe 4.Schritt)

2. Pins selektieren
   - Cursor auf Anfangspunkt setzen
   - Funktionstaste <F1> drücken und dabei mit der Maus gewünschten Selektrahmen aufziehen.

3. Zusammenfassen der selektierten Pins, Netze und Instances zu einer Gruppe.
   QuickFault> GROUP group_name

4. Ausblenden der Fehler für den nächsten Simulationslauf
   QuickFault> FAUlts TYpe group_name Both -Inhibit -Select

Anhang F – Erstellung eines Gate–Array–Layouts mit AUTOGATE und GATEGRAPH

AUTOGATE™ und GATEGRAPH™ (Mentor Graphics) sind Werkzeuge zur Erstellung und Bearbeitung eines Gate–Array–Layouts. Gearbeitet wird mit einem interaktiven Grafik–Editor, GATEGRAPH, in Verbindung mit den automatischen Plazierungs– und Verdrahtungsprogrammen, GATEPLACE und GATEROUTE. AUTOGATE (d.h. GATEPLACE und GATEROUTE) und GATEGRAPH unterstützen 2–Layer–Gate–Arrays. Bild F–1 zeigt einen typischen Design–Entwicklungsablauf.

Nachdem das Design mit NETED erstellt und evtl. durch QUICKSIM getestet wurde, kann mit Hilfe von AUTOGATE und GATEGRAPH das Layout erzeugt werden.

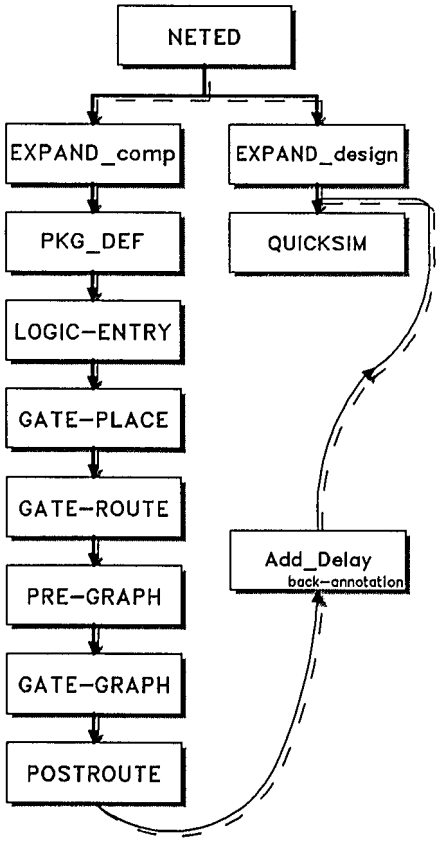


Bild F–1: Entwicklungsablauf mit Gate–Array–Layout

Die folgenden Schritte verwenden ausschließlich Befehle und Bibliotheken, die AMS (Austria Mikro Systeme) zur Verfügung stellt.

1. Schritt: Expandieren des Designs.

$ **ams_expand_comp** design_name AMS_library

design_name Pfadname des Designdirectorys;

AMS_library AMS–Gate–Array–Library, die für dieses Design verwendet wurde. (z.B. ga_2dm)

Beschreibung:

Aus der Design–Datenbasis werden die für das Layout wichtigen Properties extrahiert.

Beispiel:

```
$ ams_expand_comp test_design ga_2dm

Calling up the Mentor Expand program...

EXPAND V7.0_1.7  Monday, March 6, 1989   9:25:36 am (PST)
  Incorporating: DFA V7.0_1.3  Saturday, March 4, 1989   2:16:05 pm (PST)

Copyright (c) Mentor Graphics Corporation, 1982, All Rights Reserved.
                UNPUBLISHED, LICENSED SOFTWARE.
    CONFIDENTIAL AND PROPRIETARY INFORMATION WHICH IS THE
  PROPERTY OF MENTOR GRAPHICS CORPORATION OR ITS LICENSORS.

    EXPAND test_design comp
    PROPerty Pin pintype load size krise kfall phy_pin
    PROPerty Net pintype prio wtype vtype
    PROPerty Instance comp split ref place seed area sc_io ga
    PROPerty Instance gc inst_tid cell_type
    PRIMitive comp ''
    RUN -List
#   Processing "/user/ams/ga_2dm/cells/ib015":
#   Processing "/user/ams/ga_2dm/cells/iill":
#   ..........
#   Processing "/user/ams/ga_2dm/cells/no04":
#   Processing "/user/ams/ga_2dm/cells/ob015":
#
#   No errors, no warnings, 54 instances generated.
    BYE
#   Writing file "/user/hans/gate/demo/test_design/comp.erel_$2"
```

2. Schritt: Zuordnung der I/O-Signal-Namen zu den Pins.

$ **ams_pkg_def** design_name AMS_library array Gehäuse

design_name	Pfadname des Designdirectorys;
AMS_library	AMS-Gate-Array-Library, die für dieses Design verwendet wurde; (z.B. ga_2dm)
array	Verwendete Gate-Array-Größe (z.B. ams1k, ams2k, ams4k, ..);
Gehäuse	Verwendetes Gehäuse; gültig für Array-Größe (z.B. pkg_24_dip).

Beschreibung:

AMS_PKG_DEF ist ein Werkzeug, mit dem die Design-I/O-Signale den Pins des Gehäuses zugeordnet werden können. Zu diesem Zweck werden die grafischen Möglichkeiten von NETED verwendet (siehe auch Beispiel).

- I/O-Signal-Namen mit der linken Maustaste anklicken;
- Cursor auf gewünschten IONET-Text eines Pins setzen und RETURN-Taste drücken;
- analog mit der Spannungsversorgung verfahren;
- Abschließen: Durch Anklicken von $_EXIT.

AMS_PKG_DEF erzeugt das DDF (Design-Definition-File) design_name/place.ddf in welchem die Positionen der I/O-Puffer abgelegt sind (wichtig für GATEPLACE und GATEROUTE).

Beispiel:

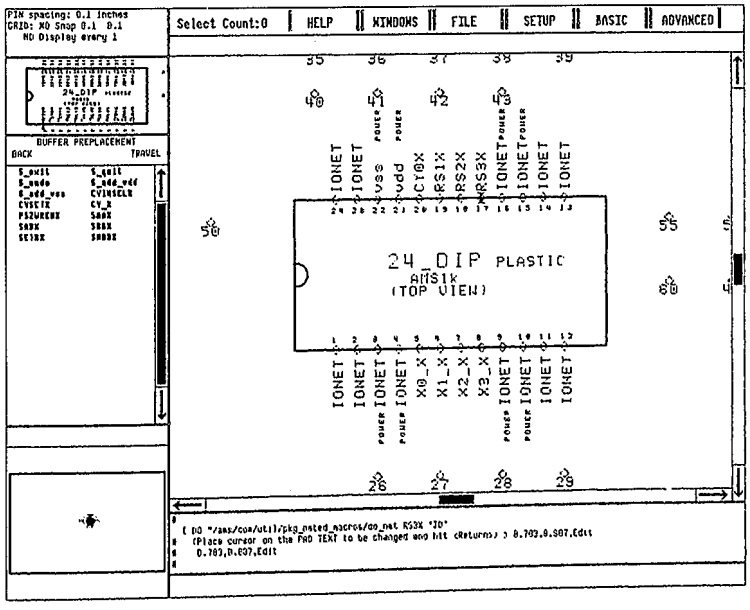
```
$ ams_pkg_def test_design ga_2dm ams1k pkg_24_dip

 Now starting  PACKAGE DEFINITION  utility...

 Extracting IO data from test_design...
```

```
*******************************************************************************
* AMS Mentor Design Kit Utility: PLACE_NETLIST_1          1.00_7.0/JAN-08-90  *
*******************************************************************************
Copyright (c) Mentor Graphics Corporation, 1982, All Rights Reserved.
              UNPUBLISHED, LICENSED SOFTWARE.
    CONFIDENTIAL AND PROPRIETARY INFORMATION WHICH IS THE
  PROPERTY OF MENTOR GRAPHICS CORPORATION OR ITS LICENSORS.

 Going into NETED to determine pinout...
```




```
 Extracting package preplacement information for test_design...

 Checking for errors...
                none.
 Creating output files...
*******************************************************************************
* AMS Mentor Design Kit Utility: PLACE_NETLIST_2          1.00_7.0/JAN-08-90  *
*******************************************************************************
Copyright (c) Mentor Graphics Corporation, 1982, All Rights Reserved.
              UNPUBLISHED, LICENSED SOFTWARE.
    CONFIDENTIAL AND PROPRIETARY INFORMATION WHICH IS THE
  PROPERTY OF MENTOR GRAPHICS CORPORATION OR ITS LICENSORS.

    DDF Output may be found in -  test_design/place.ddf
    PINOUT Report may be found in -  test_design/pinout.report

 *** AMS Package Definition Completed ***
```

3. Schritt: Starten von AUTOGATE als Remote–Job auf einer anderen Workstation (wenn AUTOGATE auf der Arbeitsstation nicht autorisiert ist).

$ crp –on //WSx –me

4. Schritt: Bereitstellen des physikalischen Designs.

$ **ams_layout logic_entry** design_name AMS_library array

design_name Pfadname des Designdirectorys;

AMS_library AMS–Gate–Array–Library, die für dieses Design verwendet wurde; (z.B. ga_2dm)

array Verwendete Gate–Array–Größe (z.B. ams1k, ams2k, ams4k, ..).

Beschreibung:

Das LOGIC_ENTRY Kommando erzeugt aus einem logischen Design (erzeugt mit NETED) das physikalische Design. Dabei wird geprüft, ob der Entwurf in das gewählte Array paßt (das Array ams1k ist im unteren Beispiel nur zu 33.2% belegt).

Beispiel:

```
ams_layout logic_entry  test_design ga_2dm ams1k

Now calling up the Mentor LOGIC_ENTRY program...

LOGIC_ENTRY v7.0_1.32  Monday, September 11, 1989   2:17:01 pm (PDT)
     LOGIC_ENTRY test_design ~/ams/ga_2dm/ams1k
NETLIST MODULE V7.0_1.4  Monday, January 30, 1989   2:49:59 pm (PST)

Copyright (c) Mentor Graphics Corporation, 1982, All Rights Reserved.
                UNPUBLISHED, LICENSED SOFTWARE.
    CONFIDENTIAL AND PROPRIETARY INFORMATION WHICH IS THE
  PROPERTY OF MENTOR GRAPHICS CORPORATION OR ITS LICENSORS.

PHYSICAL_DESIGN_FILE_CREATION completed successfully
Processing the package preplacement...
PACKAGE_PREPLACEMENT completed successfully
Design file report is in test_design/design.rpt
 Required sites            =   429 out of  1293 ( 33.2%)
 Required internal sites   =   412 out of  1232 ( 33.4%)
 Required buffer sites     =    17 out of    61 ( 27.9%)
 ams_layout LOGIC_ENTRY  Completed - please check any warnings or errors.
```

5. Schritt: Automatisches Plazieren.

$ **ams_layout gateplace** design_name AMS_library array

design_name Pfadname des Designdirectorys;

AMS_library AMS–Gate–Array–Library, die für dieses Design verwendet wurde; (z.B. ga_2dm)

array Verwendete Gate–Array–Größe (z.B. ams1k, ams2k, ams4k, ..).

Beschreibung:

GATEPLACE führt eine automatische Plazierung von Makrozellen durch, deren Plazierung weder durch Properties in NETED, noch durch das Design–Definition–File, oder durch eine manuelle Teilplazierung vorgeschrieben ist.

Beispiel:

```
$ ams_layout gateplace test_design ga_2dm ams1k

Calling up the Mentor  gateplace  program...

GATEPLACE v7.0_1.9  Wednesday, August 30, 1989   10:35:11 am (PDT)
     GATEPLACE test_design ~/ams/ga_2dm/ams1k
GATEPLACE_PHASE_1 completed successfully
GATEPLACE_PHASE_2 completed successfully
GATEPLACE_PHASE_3 completed successfully

AMS_LAYOUT  gateplace  Completed - please check any warnings or errors.
```

6. Schritt: Automatisches Routen.

$ **ams_layout gateroute** design_name AMS_library array

design_name Pfadname des Designdirectorys;

AMS_library AMS–Gate–Array–Library, die für dieses Design verwendet wurde; (z.B. ga_2dm)

array Verwendete Gate–Array–Größe (z.B. ams1k, ams2k, ams4k, ..).

Beschreibung:

GATEROUTE führt eine automatische Verdrahtung von Makrozellen durch, deren Verdrahtung nicht durch eine manuelle Teilverdrahtung vorgeschrieben ist. In dem durch GATEROUTE erzeugten Log–File "<design_name>_verify" kann der Routing–Grad entnommen werden.

Beispiel:

```
$ ams_layout gateroute  test_design ga_2dm ams1k

Calling up the Mentor  gateroute  program...

GATEROUTE v7.0_1.6  Friday, September 1, 1989   4:35:37 pm (PDT)
     GATEROUTE test_design ß/ams/ga_2dm/ams1k
GATEROUTE_INPUT completed successfully
SECONDARY_SWAP completed successfully
CAPACITY_COUNTER completed successfully
GLOBAL_ROUTING completed successfully
PIN_SWAP completed successfully
VERTICAL_TRACK completed successfully
CONTINUITY completed successfully
STREET_ROUTER completed successfully
MAZE_RUNNER completed successfully
DANGLING_REMOVAL completed successfully
OVERFLOW_RECOMP completed successfully
MAZE_RUNNER completed successfully
RIP_AND_RETRY completed successfully
DANGLING_REMOVAL completed successfully
VERIFICATION completed successfully
GATEROUTE completed; wiring results are in file test_design_verify

AMS_LAYOUT  gateroute  Completed - please check any warnings or errors.
```

7. Schritt: Abschluß des Remote–Jobs für AUTOGATE.

Nachdem die AUTOGATE–Werkzeuge erfolgreich durchlaufen wurden, kann mit GATEGRAPH weitergearbeitet werden. Zu diesem Zweck muß der vorher durch crp (create_process) gestartete Remote–Process beendet werden.

Bewegen Sie den Cursor zur Shell–Eingabezeile und drücken Sie

<CTRL&Z>

8. Schritt: Erzeugen eines GATEGRAPH–Workfiles.

$ **ams_layout pregraph** design_name AMS_library array

design_name Pfadname des Designdirectorys;

AMS_library AMS–Gate–Array–Library, die für dieses Design verwendet wurde; (z.B. ga_2dm)

array Verwendete Gate–Array–Größe (z.B. ams1k, ams2k, ams4k, ..).

Beschreibung:

Die von GATEPLACE und GATEGRAPH erzeugten Daten–Files können von GATEGRAPH nicht gelesen werden. Ein Pre–Processor (PREGRAPH) erstellt ein Arbeitsfile für GATEGRAPH.

9. Schritt: Aufruf von GATEGRAPH: Manuelles Plazieren und Routen.

$ **ams_layout gategraph** design_name AMS_library array

design_name Pfadname des Designdirectorys;

AMS_library AMS–Gate–Array–Library, die für dieses Design verwendet wurde; (z.B. ga_2dm)

array Verwendete Gate–Array–Größe (z.B. ams1k, ams2k, ams4k, ..).

Beschreibung:

GATEGRAPH ist ein interaktives Layoutprogramm zur Plazierung von Makrozellen und Verlegung von Netzen.

10. Schritt: Bearbeitung eines Gate–Array–Layouts mit GATEGRAPH.

Bild F–2 zeigt den Bildschirmaufbau von GATEGRAPH. Mit GATEGRAPH kann von Hand ein Gate–Array–Layout erstellt oder ein von AUTOGATE erzeugtes Gate–Array–Layout nachbearbeitet werden. Die nachstehenden Ausführungen sollen die wichtigsten Editiermöglichkeiten von GATEGRAPH kurz erläutern.

GATEGRAPH ist so umfangreich, daß eine ganauere Beschreibung aller Funktionen und Befehle im Rahmen dieser Kurzanleitung nicht möglich ist. Darum wird an Hand eines kleinen Beispiels das (Um–)Plazieren einer Makrozelle und das Routen eines Netzes vorgeführt.

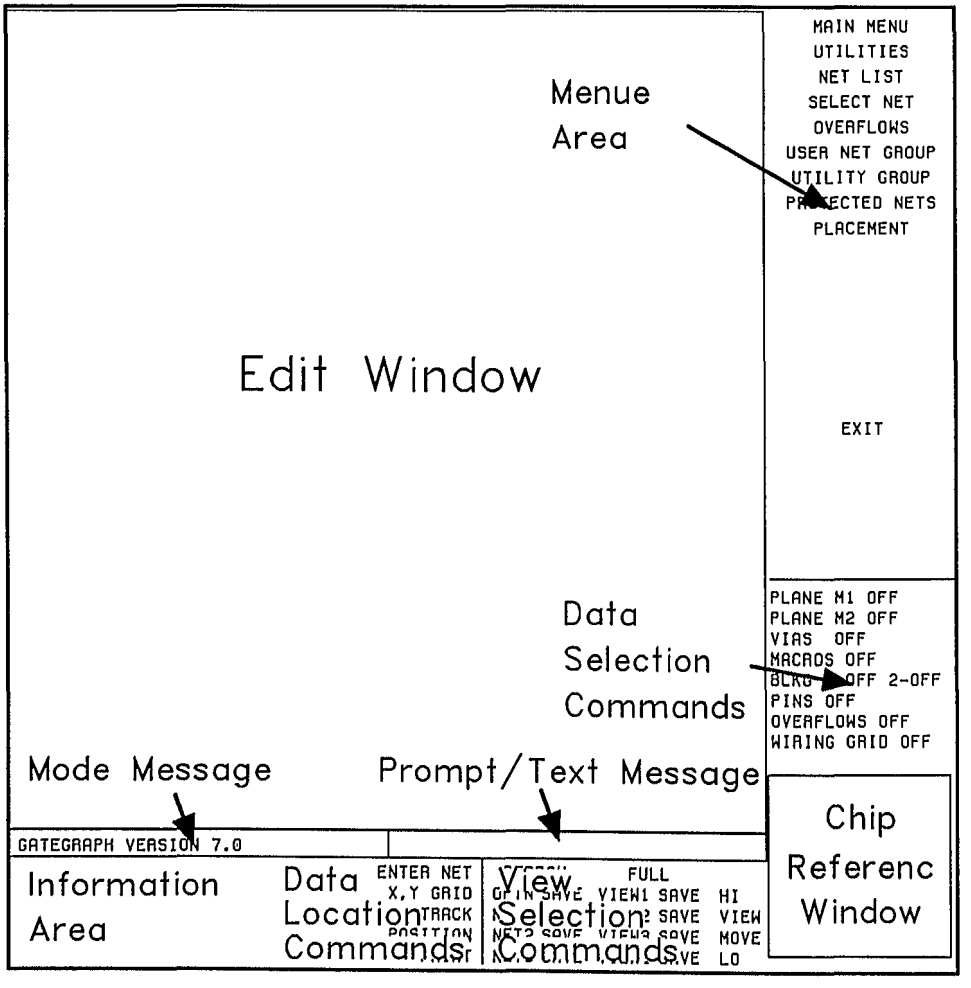


Bild F–2: Bildschirmaufbau und Fenster–Aufteilung in GATEGRAPH

Zunächst einige Anmerkungen zur Bedeutung der wichtigsten Fenster (Windows) von GATEGRAPH:

MENUE AREA–Window: In diesem Fenster werden die verschiedenen Menue–Punkte von GATEGRAPH ausgewählt, z.B. SELECT NET zum Netze selektieren; OVERFLOWS zum Sichtbarmachen nicht gerouteter Netze; PLACEMENT zum Plazieren der Makrozellen u.a. .

DATA SELECTION COMMAND–Window: Mit Hilfe der Kommandos dieses Fensters kann die Darstellungsvielfalt im *EDIT*–Window beeinflußt werden. Es können die Verdrahtungsebenen, Durchkontaktierungen, Pins, u.s.w. sichtbar oder unsichtbar gemacht werden.

INFORMATION AREA: Hier werden Makro– bzw. Netzinformationen angezeigt.

VIEW SELECTION COMMAND–Window: In diesem Fenster können Netze oder Bildschirmausschnitte abgespeichert und bei Bedarf durch VIEW wieder angezeigt werden.

Beispiel:

Nach dem Aufruf von GATEGRAPH erscheint das Arbeitsfenster (siehe Bild F–2). Durch Anklicken von MACROS, PLANE M1 und PLANE M2 im *DATA SELECTION COMMAND*–Window wird die Plazierung und Verdrahtung der einzelnen Makrozellen auf dem Gate–Array sichtbar.
Als ersten Schritt soll eine Makrozelle umplaziert werden. Dazu ist im *MENUE AREA*–Window der Menuepunkt PLACEMENT anzuklicken.

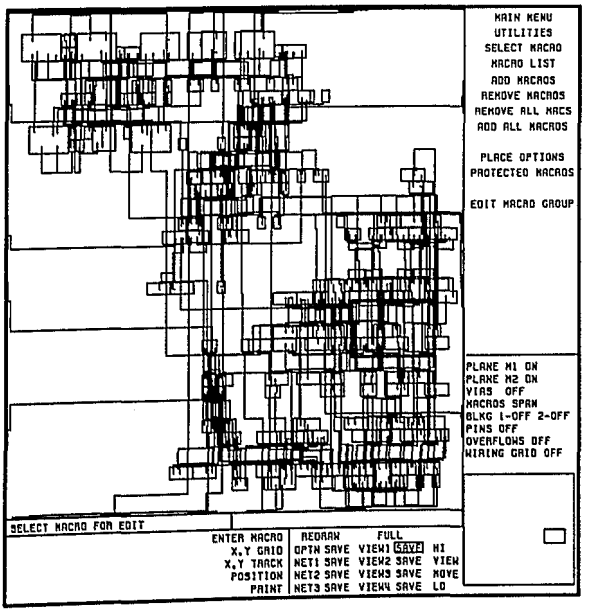


Bild F–3: Gate–Array–Layout als Ergebnis von AUTOGATE; Bearbeitung mit GATEGRAPH

Allgemeines:

Es besteht die Möglichkeit Makrozellen und Netze an Hand ihrer Kennung (Handle: z.B. I$57,N$345,..) zu selektieren. Dadurch kann der Anwender kritische Netze und/oder Makrozellen auffinden und manuell bearbeiten.

Netze auswählen:

NET LIST
Netz–Kennung anklicken
RETURN TO MAIN MENU
USER NET GROUP

Makrozellen auswählen:

PLACEMENT
MACRO LIST
Makro–Kennung anklicken
RETURN TO MAIN MENU
EDIT MACRO GROUP

Bildausschnitt verändern:

Mit <F8> Anfangspunkt setzen, Rahmen mit der Maus aufziehen und linke Maustaste drücken (Bild F–3).

Makrozelle verschieben:

SELECT MACRO (Bild F–3); Makrozelle anklicken;
MOVE MACRO (Bild F–4); anschließend ist die Makrozelle mit der Maus zu verschieben und durch anklicken der linken Maustaste zu plazieren (Bild F–5).

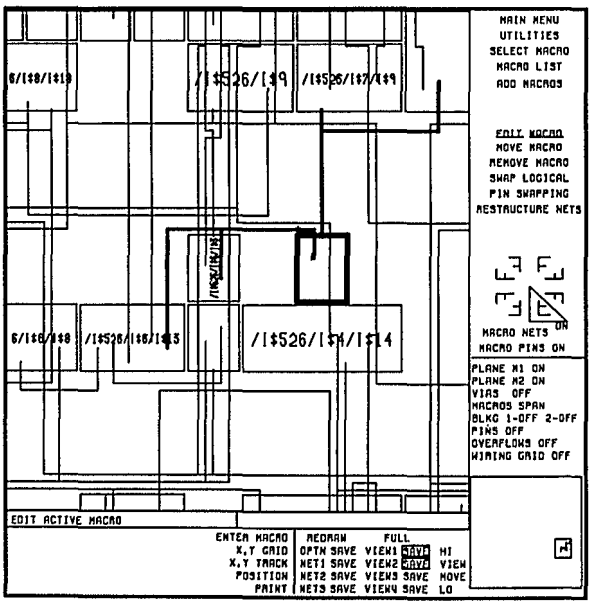


Bild F–4: Makrozelle verschieben; Schritt 1

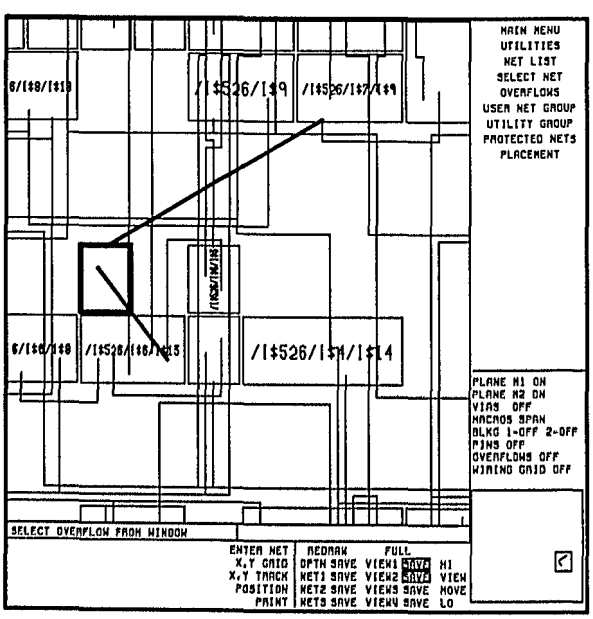


Bild F–5: Makrozelle verschieben; Schritt 2

Netze routen:

Auf dem Gatte–Array können nicht routebare Bereiche der einzelnen Lagen sichtbar gemacht werden; hierzu ist im *DATA–SELECTION–COMMAND*–Window BLKG_1 und BLKG_2 anzuklicken (Bild F–6).

Selektieren eines nicht gerouteten Netzes:

Im *MENUE–AREA*–Window MAIN MENU anklicken, anschließend mit OVERFLOWS die nicht gerouteten Netze anzeigen lassen. Den Cursor auf dem gewünschten Netz plazieren und linke Maustaste anklicken; das Netz ist nun selektiert. Durch nochmaliges Anklicken des Netzes gelangt man ins Netz–Edit–Menue und kann mit dem manuellen Routen beginnen (Bild F–7).

Netz–Edit–Menue:

A AUTO B: Routet automatisch von A nach B oder umgekehrt.

A ↕ B: Zeigt Routemöglichkeiten von A bzw. B in vertikaler Richtung.

A ↔ B: Zeigt Routemöglichkeiten von A bzw. B in horizontaler Richtung (Bild F–8).

A ADD VIA B: Setzt eine Durchkontaktierung an die Position von A oder B. Dadurch kann die Arbeitsebene gewechselt werden.

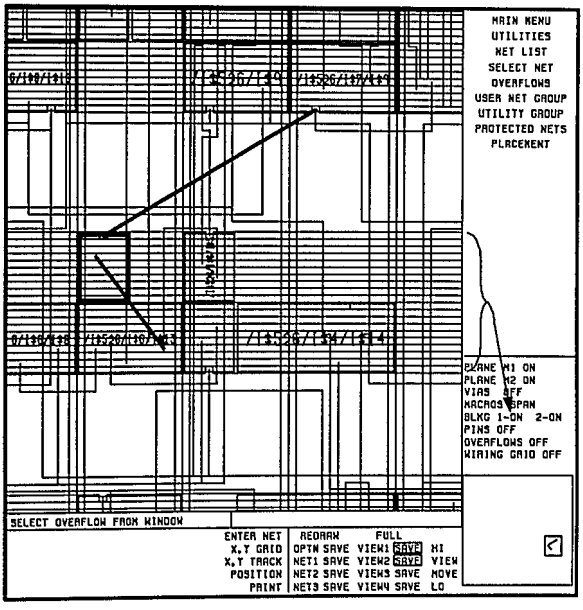


Bild F–6: Zum Selektieren nicht gerouteter Netze

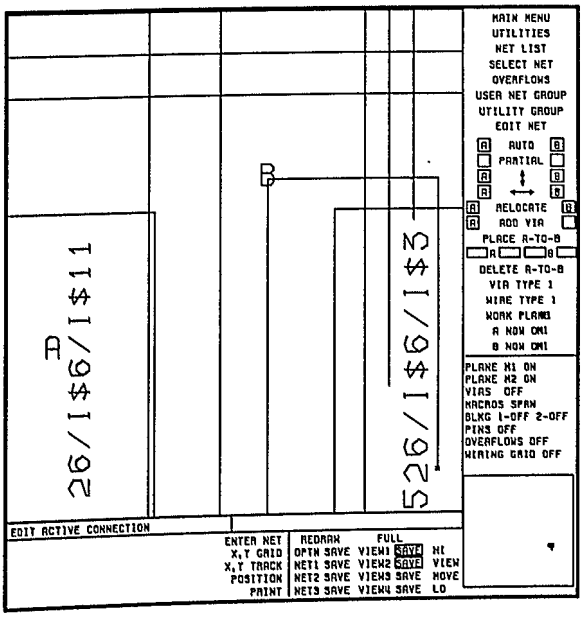


Bild F–7: Zum Routen von Netzen

← A → ← B → : Hiermit kann die Position von A u. B entlang der aktiven Verbindung verschoben werden. So wird z.B. durch Anklicken von ←A der Punkt A von B wegverschoben.

DELETE A–TO–B: Es kann der durch obige Funktion festgelegte Netzabschnitt zwischen A und B gelöscht werden.

Mit A ↕, ↔ B wird der horizontale bzw. vertikale freie Netzweg als Rechteck angezeigt. Durch Anklicken des gewünschten Leitungsendpunktes im Rechteck wird bis zu dieser Stelle geroutet. Erforderlichenfalls durch Anbringen von Durchkontaktierungen (VIAS), sollte es möglich sein bis zum Endpunkt zu routen (Bild F–8 u. Bild F–9).

Abschluß von GATEGRAPH:

MAIN MENU
EXIT

WORKFILE:
Im Workfile wird der aktuelle Zustand der Arbeitssitzung abgelegt.

DESIGNFILE:
Daten für Backannotation (neue Leitungslängen und Positionen der Makrozellen werden im Designfile abgelegt).

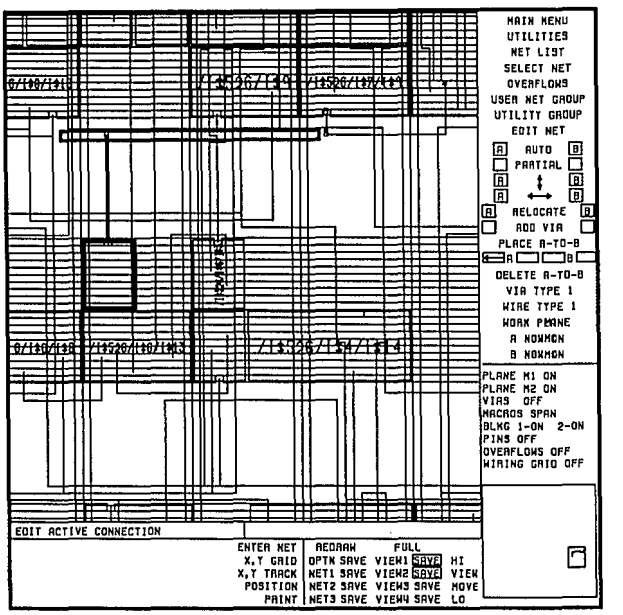


Bild F–8: Zum Routen von Netzen

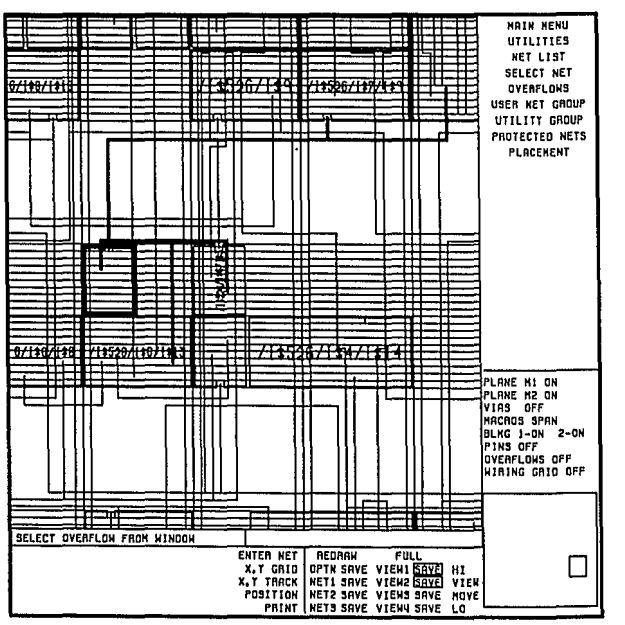


Bild F–9: Zum Routen von Netzen

Anhang G – Entwicklungsschritte für Gate–Array–Design mit Beispiel

Die Entwicklung einer (digitalen) Schaltung und ihre Implementierung in Gate–Array–Technologie erfolgt in mehreren Schritten:

1. Schritt: Spezifikation.

In der Spezifikation wird der Leistungsumfang der Schaltung, ihr Zeitverhalten, Gehäuseform und Pinbelegung festgelegt. Bei der Festlegung der Pinbelegung sind bereits Entwurfsvorschriften des Halbleiterherstellers zu berücksichtigen. In diesen *Design–Rules* ist vorgeschrieben, welche Gehäuse–Pins für Input–/Output–Anwendungen bzw. für Stromversorgungszwecke verfügbar sind. Insbesondere gibt es Vorschriften für die Zahl und Lage der Stromversorgungsanschlüsse in Abhängigkeit der Input–/Output–Verhältnisse der Schaltung, sowie möglicherweise reservierte Pins für Testzwecke. *Die Vollständigkeit und Korrektheit der Spezifikation ist von entscheidender Bedeutung für ein solches Entwicklungsvorhaben, da Nachbesserungen nicht möglich sind!*

Beispiel: Es ist ein elektronischer Leistungssensor zu entwerfen. Der Leistungssensor enthält einen digitalen Multiplikator zur Multiplikation der Signale zweier Analogkanäle. Je ein Vorverstärker bringt das Sensorsignal in den Aussteuerbereich 0 bis 4V des A/D–Wandlers. Der A/D–Wandler soll mit 8 Bit auflösen. Die daraus gewonnenen 8–Bit–Datenwörter sind digital zu multiplizieren. Das Ergebnis der Multiplikation ist über einen 16–Bit–Datenbus an die nachfolgende Signalverarbeitungseinheit abzugeben. Mit dem Steuersignal SAD wird die A/D–Wandlung initialisiert; mit START wird die Multiplikation ausgelöst. Das Übergabesignal CC meldet der Multiplikationsschaltung die Bereitstellung der Datenwörter. READY kennzeichnet die Verfügbarkeit des Ergebnisses (Bild G–1).

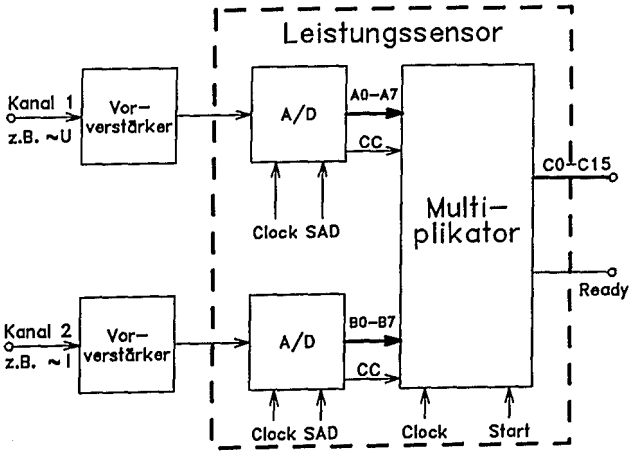


Bild G–1: Blockschaltbild des Leistungssensors

2. Schritt: Blockschaltbild und Logiksynthese von Teilfunktionen.

Da Schaltungen, die auf einem Gate–Array implementiert werden, heutzutage komplex sind und typischerweise mehrere tausend Gatter umfassen, ist eine sorgfältige Strukturierung der Schaltung erforderlich. Bereits in diesem Stadium ist die *Testbarkeit* der Schaltung zu berücksichtigen. die Zerlegung erfolgt je nach Komplexität nach der Methode der *hierarchischen sukzessiven Verfeinerung*, d.h. es wird erst eine Zerlegung in grobe Funktionsblöcke vorgenommen und deren Zusammenwirken und Verschaltung festgelegt. Jeder dieser Blöcke wird dann seinerseits weiter in feinere Funktionsblöcke zergliedert, bis schaltungstechnisch übersichtliche und wohlspezifizierte Teilfunktionen festgelegt sind.

Verfeinerung des Entwurfs am Beispiel des Multiplikators: Darstellung des Entwurfs auf Registertransferebene, mit Datenfluß zwischen den Funktionsblöcken. Die Steuerleitungen regeln den Datentransfer (Bild G–2).

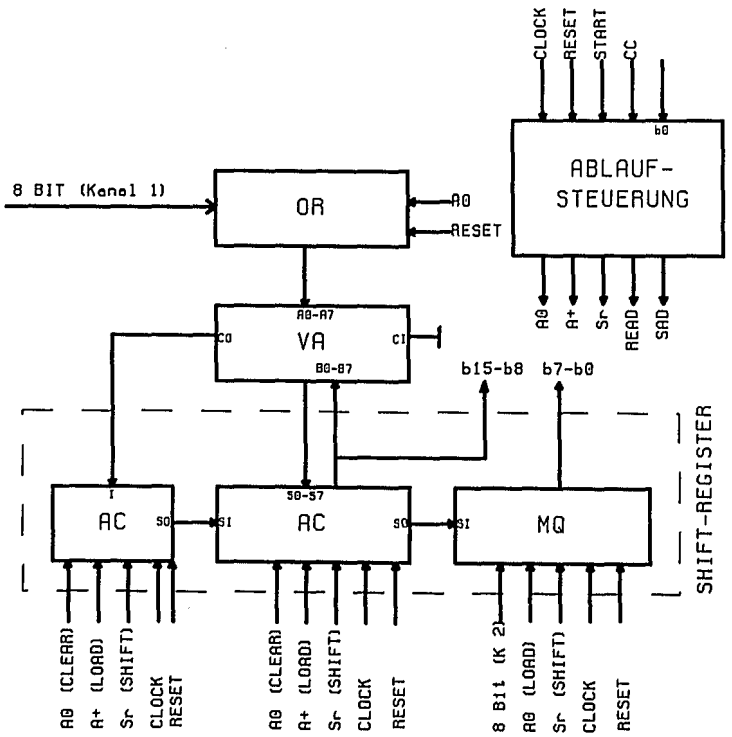


Bild G–2: Blockschaltbild des Multiplikators

Logiksynthese von Teilfunktionen: Verwendung von Logiksynthesewerkzeugen für den Entwurf von Teilfunktionen. Zustandsautomaten werden mit PLDSynthesis oder mit Log/iC entworfen. Die Ablaufsteuerung soll als Zustandsautomat die Steuersignale für den Datentransfer des Multiplikators erzeugen (Bild G-3).

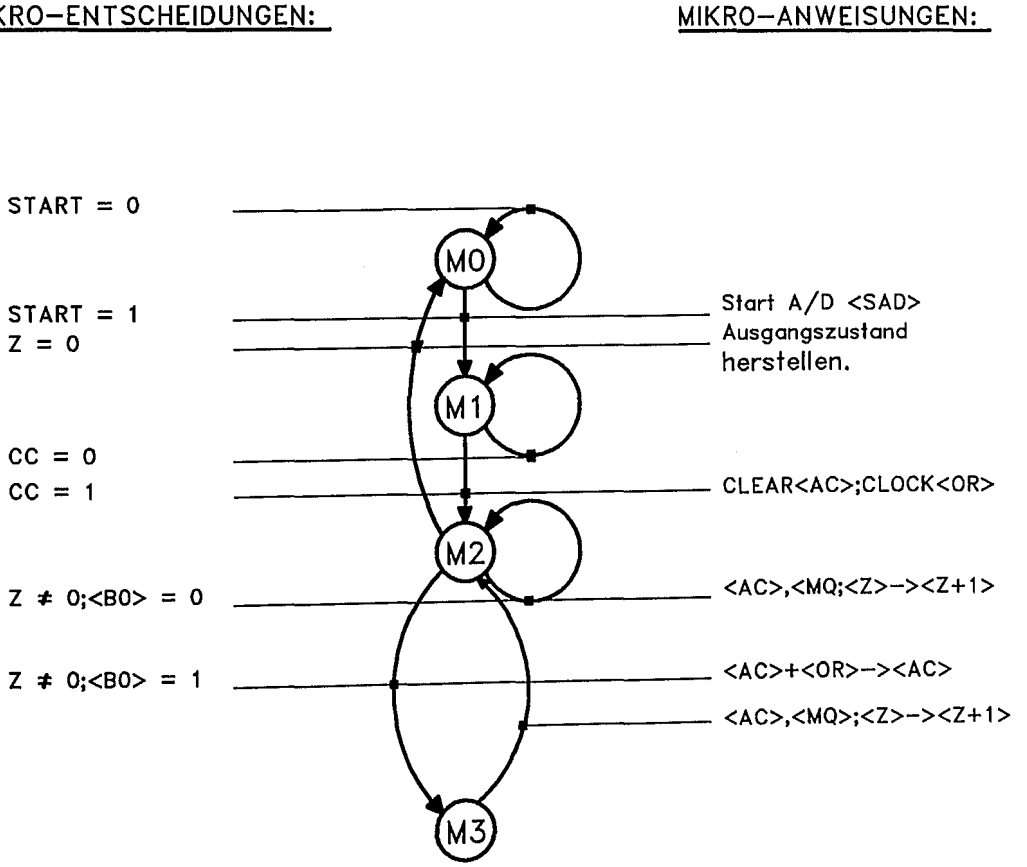


Bild G–3: Zur Logiksynthese der Ablaufsteuerung

3. Schritt: Technologieauswahl.

Nach der Zerlegung der Schaltung in schaltungstechnisch implementierbare Funktionsblöcke , erfolgt eine Abschätzung des *Zell– oder Gatterbedarfs* unter Zugrundelegung von Makrozellen–Datenbüchern der Halbleiterhersteller. Die Geschwindigkeits–, Ausgangstreiber– und Verlustleistungserfordernisse ergeben weitere Auswahlpunkte. Weitere Auswahlkriterien sind Verfügbarkeit und Umfang von Makrozell biblotheken für die einzusetzenden CAE/CAD–Werkzeuge.

4. Schritt: Schaltungsentwicklung.

Die Schaltungsentwicklung erfolgt grundsätzlich mit Rechnerunterstützung. Die Details des Entwicklungsgangs sind abhängig vom eingesetzten Rechner, der verfügbaren Entwicklungssoftware und vom *Design–Kit* des Halbleiterherstellers.

Bei der Entwicklung von Schaltungen, die auf Silizium (oder auch andere Halbleitermaterialien) implementiert werden sollen, sind die Besonderheitern der Implementierungstechnik zu berücksichtigen. Dies gilt auch für Gate–Array–Entwicklungen. Der Schaltungsentwickler arbeitet hier nicht mit Einzeltransistoren, sondern mit *Logik–Makrozellen* (z.B. Flipflop, Logik–Gatter, Schieberegister...) und ist deshalb versucht, die schaltungstechnischen Gewohnheiten einer *diskreten* Implementierung mit SSI–/MSI–Standardbausteinfamilien (z.B. TTL 74er Serie) anzuwenden. Dies kann zu funktionsunfähigen oder nicht testbaren Chips führen. Es ist deshalb *unbedingt erforderlich, die Entwurfsrichtlinien des Herstellers zu befolgen.* Hierzu gehört auch die Forderung nach einem *synchronen Schaltungsentwurf.*

Symbolische Darstellung des Entwurfs: Der Entwurf wird jetzt vollständig auf symbolischer Ebene mit der CAE–Workstation durch die Schaltplaneingabe **NETED/SYMED** dargestellt.

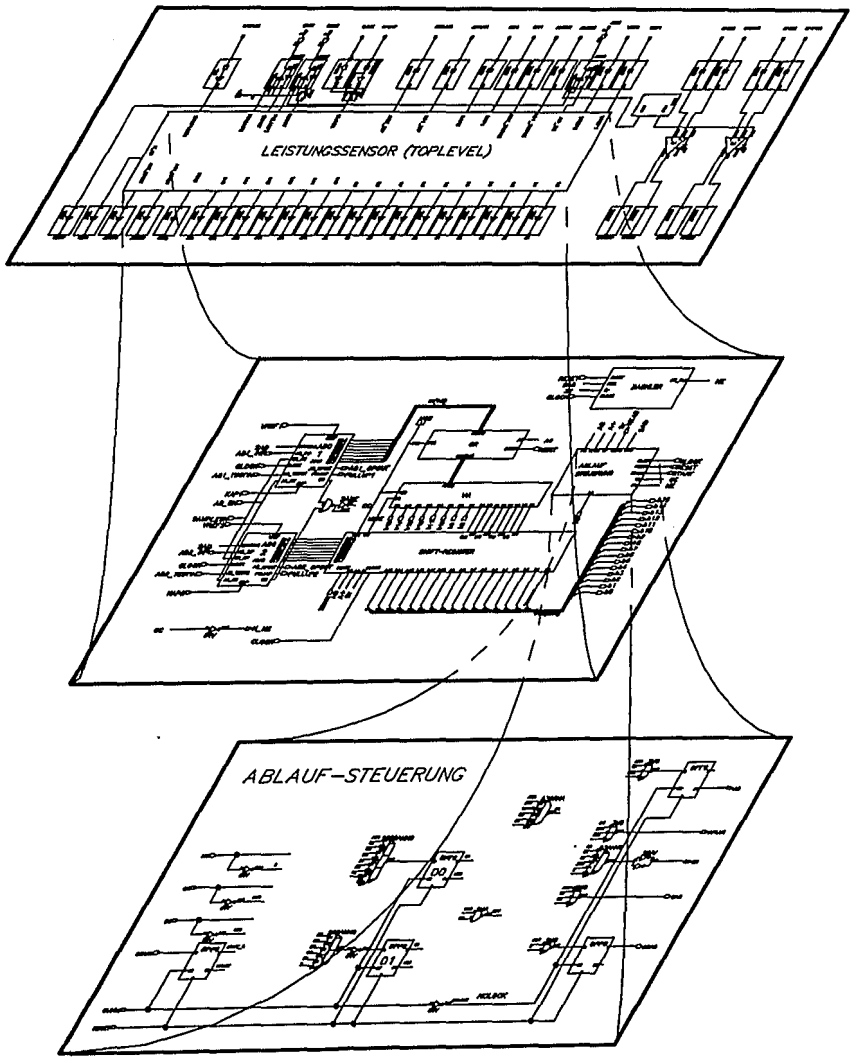


Bild G–4: Hierarchischer Schaltplan

5. Schritt: Design–Compilation.

Nach der (meistens) grafischen Schaltungseingabe wird die *formale Konsistenz* der Schaltung mit Hilfe der verfügbaren CAE–Softwarewerkzeuge überprüft. Alle gemeldeten *Fehler* müssen in der Schaltung korrigiert werden. Gemeldete *Warnungen* können auf tolerierbare Fehler (z.B. absichtlich nicht benutzter Ausgang einer Makrozelle) oder nicht tolerierbare Fehler (z.B. ein Netz, das nicht getrieben wird) sein.

6. Schritt: Funktionelle Verifikation durch Logiksimulation.

Mit Hilfe eines Logiksimulators wird die Schaltung auf ihre Funktionsfähigkeit überprüft. Als Vergleichsbasis dient die Spezifikation. Die Funktionsverifikation wird zweckmäßigerweise bottom–up, d.h. beginnend mit kleinen Teilschaltungen durchgeführt. Zur Simulation muß die zu simulierende (Teil)Schaltung mit Test–Stimuli beaufschlagt werden. Der Simulator ermittelt die Schaltungsreaktion auf die Testfälle. *Es ist deshalb notwendig, alle erforderlichen Tests zu spezifizieren und durchgeführte Tests zu protokollieren.* Der Logiksimulator deckt Logikfehler und gewisse *Timingfehler* auf.

Funktionelle Verifikation des Entwurfs: Die funktionelle Verifikation des Entwurfs erfolgt interaktiv mit dem Logiksimulator **QUICKSIM** nach vorheriger Festlegung der für den Test notwendigen Eingangssignale.

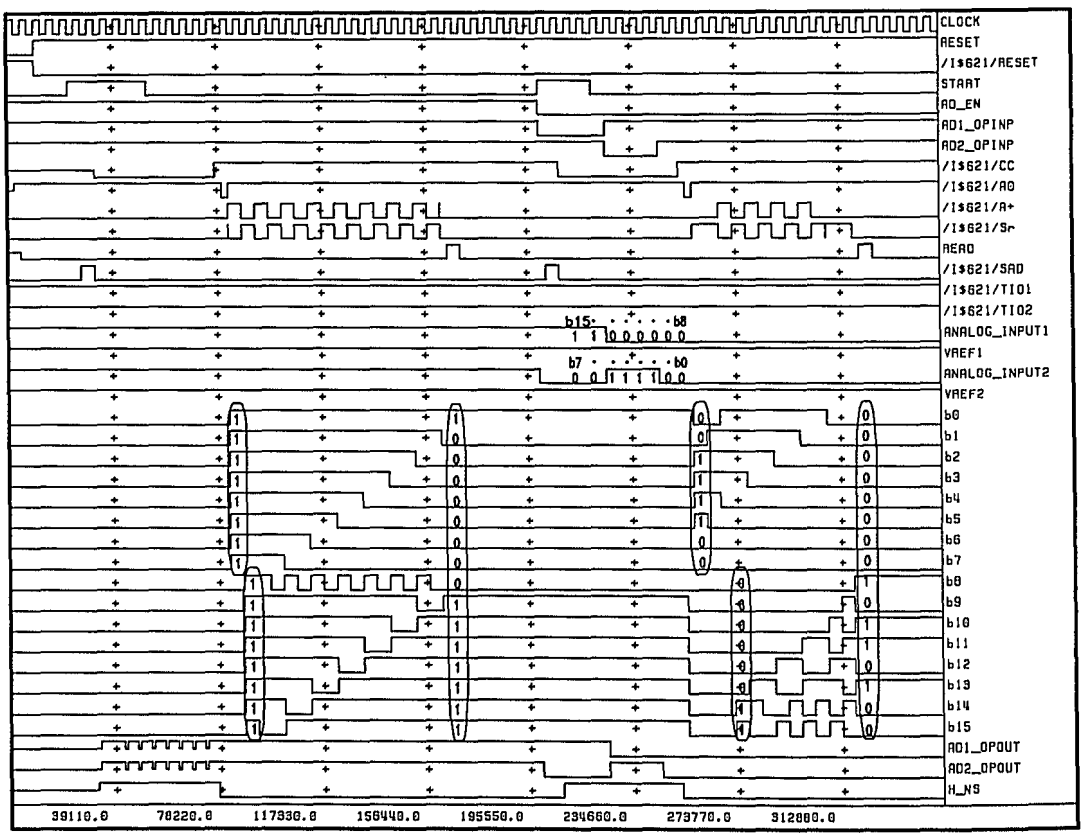


Bild G–5: Ergebnis der Logiksimulation

7. Schritt: Worst Case Timingsimulation.
Der Logiksimulator simuliert die Schaltung wahlweise unter Zugrundelegung *nominaler, minimaler* oder *maximaler* Signal– oder Gatterlaufzeiten. Mit Hilfe eines Timingsimulators läßt sich Fehlverhalten der Schaltung ermitteln, das durch die zeitliche Unsicherheit minimal...maximal auftreten kann. Die Timingsimulation ist nicht ganz unproblematisch und kann zu pessimistischen (oder auch irreführenden) Aussagen führen!

8. Schritt: Testvektorerstellung und Fehlersimulation.
Da die Fertigungsausbeute bei der Chipherstellung, insbesondere bei neuen Herstellungsprozessen, beträchtlich unterhalb von 100% liegen kann, muß jeder einzelne Chip auf Funktionfähigkeit mit Hilfe eines Testautomaten geprüft werden. Dabei kommt es lediglich darauf an festzustellen, ob ein Chip funktioniert oder nicht. Die eigentliche Ausfallursache ist für die Serienfertigung nicht wesentlich. Der Schaltungsentwickler erstellt zu diesem Zweck eine Folge von Testsignalen *(Testvektoren)* und die zu erwartenden Ausgangssignalfolgen der Schaltung. Der Testautomat beaufschlagt beim Test den Chip mit den Testvektoren und vergleicht seine Ausgangssignalfolge mit der Vergleichssignalfolge. Bei Abweichungen wird der Chip als defekt angesehen. Die Zahl der Testvektoren soll dabei minimiert werden. Die Entwicklung von Testvektoren für die Schaltung kann ein beachtlicher Anteil an der Gesamtentwicklung sein.

Verifikation der Teststimuli: Mit dem Fehlersimulator **QUICKFAULT** wird ermittelt, inwieweit das gewählte Testpattern in der Lage ist Fertigungsfehler aufzudecken.

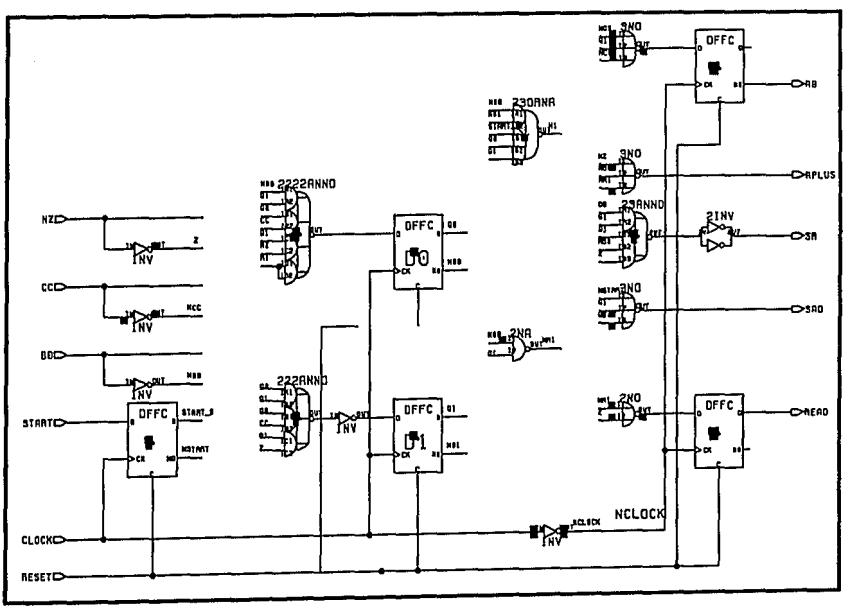


Bild G–6: Fehlersimulation der Ablaufsteuerung

9. Schritt: Erste Designübergabemöglichkeit.

Nach der Erstellung der Testvektoren kann eine Übergabe der Entwurfsdatenbasis an den Halbleiterhersteller erfolgen. Dazu wird aus der Datenbasis des Schaltplans eine Netzliste im Herstellerformat erstellt und zusammen mit den Schaltplänen, der Pinzuordnung, den Testvektoren für die funktionelle Verifikation und den Testvektoren für den Testautomaten übergeben *(Semi–validated Netlist)*. Der Hersteller verifiziert die Simulationsergebnisse des Kunden mit *seinem* Logiksimulator auf *seinem* CAD–System *(Pre–Layout–Simulation)*. Anschließend wird vom Hersteller die Plazierung der I/O–Makrozellen und der internen Logikzellen auf dem Chip festgelegt (Placement) und die Verdrahtung durchgeführt (Routing). Diese Schritte geschehen üblicherweise automatisch. Manuelle Eingriffe erfolgen nur, wenn die Schaltung *kritische Pfade* enthält, deren Signallaufzeiten durch besondere Makrozellanordnungen und Verdrahtungsführung garantiert werden müssen oder falls der *Autorouter* die Verdrahtung nicht vollständig vornehmen kann. Nach erfolgtem Layout werden aus der aktuellen Verdrahtung die Korrekturen der Signallaufzeiten errechnet und eine *Post–Layout–Simulation* der Schaltung durchgeführt. Die Ergebnisse muß der Kunde überprüfen und eine Freigabe für die Maskenherstellung und Musterfertigung bestätigen.

10. Schritt: Package–Definition.

Dieser und die folgenden Schritte sind nur erforderlich, wenn der Schaltungsentwickler auch das Chiplayout selbst durchführt.

Durch die Vorgabe der Pinbelegung eines ausgewählten Chip–Gehäuses werden Festlegungen für die Plazierung der Makrozellen getroffen *(Vorplazierung)*.

11. Schritt: Plazierung der Makrozellen.

Die Plazierung der Makrozellen auf dem Chip *(Basis–Array, Uncommitted Array)* kann grafisch interaktiv oder automatisch erfolgen. Mischformen sind möglich. Sie werden angewandt, wenn Signalpfade zeitkritisch sind *(critical path)* und durch eine interaktive Vorplazierung Toleranzen der zu erwartenden Signallaufzeiten reduziert werden sollen. Nach einer solchen Vorplazierung können die restlichen Makros durch den *Autoplacer* positioniert werden.

12. Schritt: Verdrahtung der Makrozellen.

Die Verdrahtung der Makrozellen kann grafisch interaktiv oder automatisch erfolgen. Mischformen sind möglich. Sie werden angewandt, wenn kritische Pfade in der Schaltung existieren, deren Laufzeittoleranzen eingeengt werden müssen.

13. Schritt: Gewinnung der aktuellen Signallaufzeiten.

Nach erfolgtem Layout werden aus der aktuellen Verdrahtung die *Leitungslängen* und *Kapazitätsbeläge* ermittelt und die Signallaufzeiten in der Simulationsdatenbasis aktualisiert *(Backannotation)*.

14. Schritt: Post–Layout–Simulation.

Nach erfolgter Verdrahtung und Backannotation wird die funktionelle Logiksimulation und die Timingsimulation wiederholt und die Funktionsfähigkeit der Schaltung auf dem Chip überprüft (*Postsimulation*).

15. Schritt: Übergabe der Datenbasis an den Halbleiterhersteller.

Aus dem jetzt vorliegenden Daten wird die Beschreibung des Chip–Layouts in einem vom Hersteller vorgeschriebenen Datenformat generiert und zusammen mit den Testvektoren übergeben. Der Hersteller fertigt daraus unmittelbar die Verdrahtungsmaske(n) und die Muster.

16. Schritt: Musterprüfung und Musterfreigabe.

Nach Lieferung der Muster des neuen Schaltkreises wird der Chip vom Kunden funktionell und parametrisch geprüft. Nach der Fertigungsfreigabe fertigt der Halbleiterhersteller die vereinbarte Stückzahl.

Stichwortverzeichnis

Hüthig

Adolf Auer

PLD-Handbuch

Tabellen und Daten

1990, XI, 302 S., zahlr. Abb. und Tab., kart., DM 78,—
ISBN 3-7785-1991-3
Reihe: Mikroelektronik, Band 6

Programmierbare Logik-IC haben einen festen Platz in der Schaltungsentwicklung gefunden. Mit entsprechender Literatur kann die Anwendung unterstützt werden. Fachbücher und Datenbücher sind dabei ein wichtiges Hilfsmittel jedes Entwicklers.

Das theoretische Wissen kann man sich aus den Fachbüchern erarbeiten. Sobald eine logische Schaltung vorliegt und die Wahl der geeigneten Bausteine getroffen werden muß, beginnt die Arbeit mit den Datenbüchern. Die Arbeit mit vielen und umfangreich angelegten Datenbüchern ist oft mühsam. Wünschenswert ist für diese Arbeit deshalb ein Tabellenbuch, in dem die wesentlichen Daten, wie z. B. die Architektur, Signal-Ein-/Ausgabe, Gehäuseform und Programmierbarkeit ohne Bezug auf einen Produzenten, zu finden sind. Erst in der letzten Entwicklungsphase sind die firmenspezifischen Datenbücher von Bedeutung.

Ähnlich wie für die bekannten Standard TTL-IC wird hier den Entwicklern ein Tabellenbuch für die PLD an die Hand gegeben. Um die Übersichtlichkeit zu erhöhen, wurde das Handbuch in drei Teile untergliedert:

- Teil 1 umfaßt die PAL-Architekturen und PAL-IC mit asynchronem Takt
- Teil 2 beinhaltet PLD mit programmierbaren Makrozellen
- Teil 3 beinhaltet die Eigenschaften der reprogrammierbaren PLD, wie GAL, PEEL und AGA.

Hüthig Buch Verlag
Im Weiher 10
6900 Heidelberg 1

Herbert Reichl

Hüthig

Hybridintegration

Technologie und Entwurf von Dickschichtschaltungen

2., überarb. Auflage 1988,
320 S., 258 Abb., 49 Tab., geb.,
DM 138,—
ISBN 3-7785-1665-5

Von alternativen Beschichtungsverfahren bis zur Zuverlässigkeitsberechnung wird alles Wesentliche für die Herstellung von integrierten Hybridbauelementen ausführlich und praxisnah behandelt.

Nach einem Überblick über die verschiedenen Techniken und Technologien bilden die Materialien der Dickschichttechnik, Pasten und Substrate, den ersten Schwerpunkt. Ein Beispiel verdeutlicht die Anwendung von Designregeln und das Vorgehen beim Hybridschaltungsentwurf. Eingegangen wird auch auf die Entwurfsunterstützung durch Rechner (CAD). Viele praktische Hinweise finden sich in der Darstellung technologischer Teilschritte wie Masken- und Siebherstellung sowie Schichterzeugung. Einer umfassenden Zusammenstellung von Bauelementen für Hybridschaltungen folgt die Beschreibung der Montage und Lötverfahren. Die Anwendung ungehäuster Halbleiterbausteine wird ebenso ausführlich diskutiert wie die Erfahrungen mit den unterschiedlichen Bondverfahren. Die Behandlung der Gehäusungs- und Passivierungsmethoden vervollständigt das zur Hybridintegration erforderliche Know-how.

Dem Ingenieur und Techniker wird damit eine solide Basis für die Anwendung zuverlässiger Integrationsmethoden und neuer Aufbau- und Verbindungstechniken mikroelektronischer Schaltungen geboten.

Hüthig Buch Verlag
Im Weiher 10
6900 Heidelberg 1

Hüthig

Adolf Auer

Programmierbare Logik-IC

Eigenschaften, Anwendung, Programmierung

1990, VIII, 198 S., zahlr. Abb.,
kart., DM 38,—
ISBN 3-7785-1910-7

Reihe: Mikroelektronik Band 3

Programmierbare Logik IC, die unter der englischen Abkürzung PLD bekannt sind, ermöglichen nicht nur einen optimalen Schaltungsaufbau sondern auch dessen Entwicklung mit Hilfe von Rechnern. Durch Programmieren eines Sicherheitsbits wird das direkte Auslesen der Funktion verhindert. So ist auch ein Schutz gegen unerlaubtes Kopieren einer Schaltung sichergestellt.

Im ersten Teil des Buches werden PLD aus der Sicht der Architekturen klassifiziert. Dadurch wird dem Leser der Einstieg in die PLD-Anwendung erleichtert.

Die weiteren Kapitel behandeln neben den unterschiedlichen Programmierungen wie Fuse-Link, EPROM, etc. für die Schaltungsentwicklung wichtige Themen wie Entwicklungsablauf, Hilfsmittel, und Anwendungsbeispiele sowie schaltungstechnische Eigenschaften der PLD.

Ein kurzes Lexikon der ASIC am Ende des Buches ist für die Arbeit mit englischen Datenbüchern sehr nützlich.

Hüthig Buch Verlag
Im Weiher 10
6900 Heidelberg 1